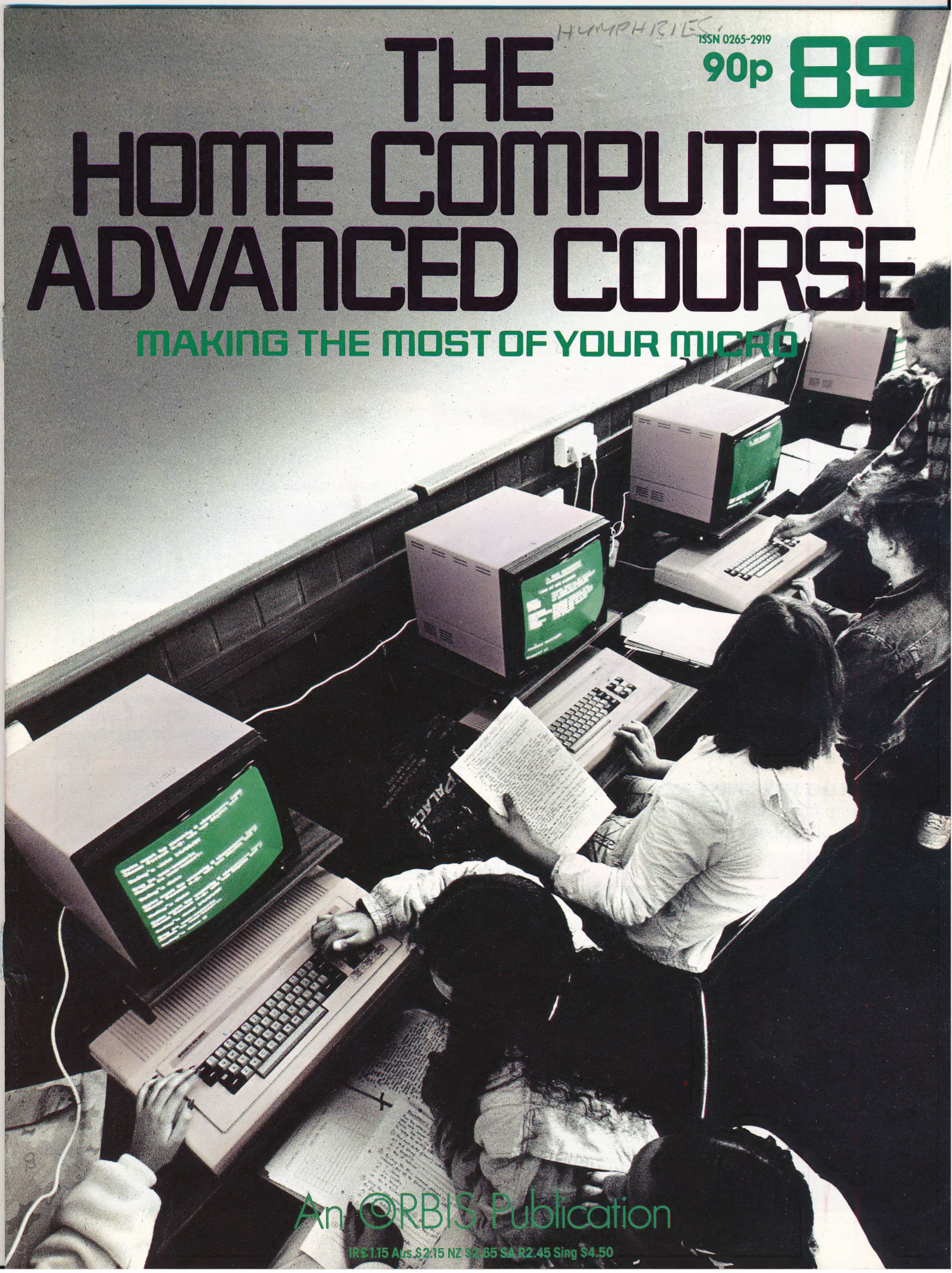
HUMPHRIES

ISSN 0265-2919

90p 89

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

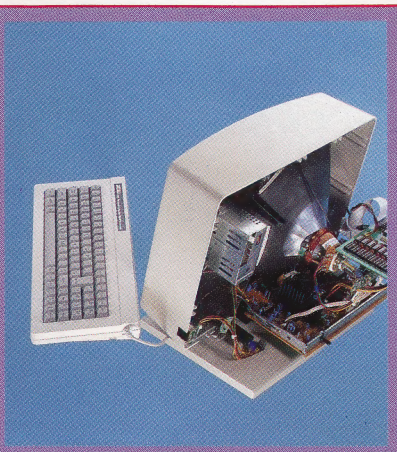
IR£1.15 Aus.$2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

Next Week

● Many people who have developed an interest in computing at home wish to channel their interest into a career. We begin a series which looks at the great variety of jobs available that involve computers.

● Until recently, dedicated word processors have generally cost well over £1,000. But all has changed with the arrival of the £400 Amstrad PCW 8256. We put the machine through its paces.



QUIZ

1) What is a 'turnkey' system?
2) If a program is written to run on a particular computer, why does it need to be 'installed'?
3) What happens when the Spectrum's ULA wishes to access the 16-Kbyte video area?

Answers To Last Week's Quiz

1) MS-DOS version 2.0 introduced hierarchical directories, while version 3.0 included multi-user support.
2) ADDA adds the contents of a memory location to an address register, while ADDI adds immediate data to an address.
3) An intelligent terminal has on-board processing power, whereas a dumb terminal has no such facility.
4) The MacWrite 'clipboard' is an area of memory set aside for the storage of blocks, which can be viewed by the user.

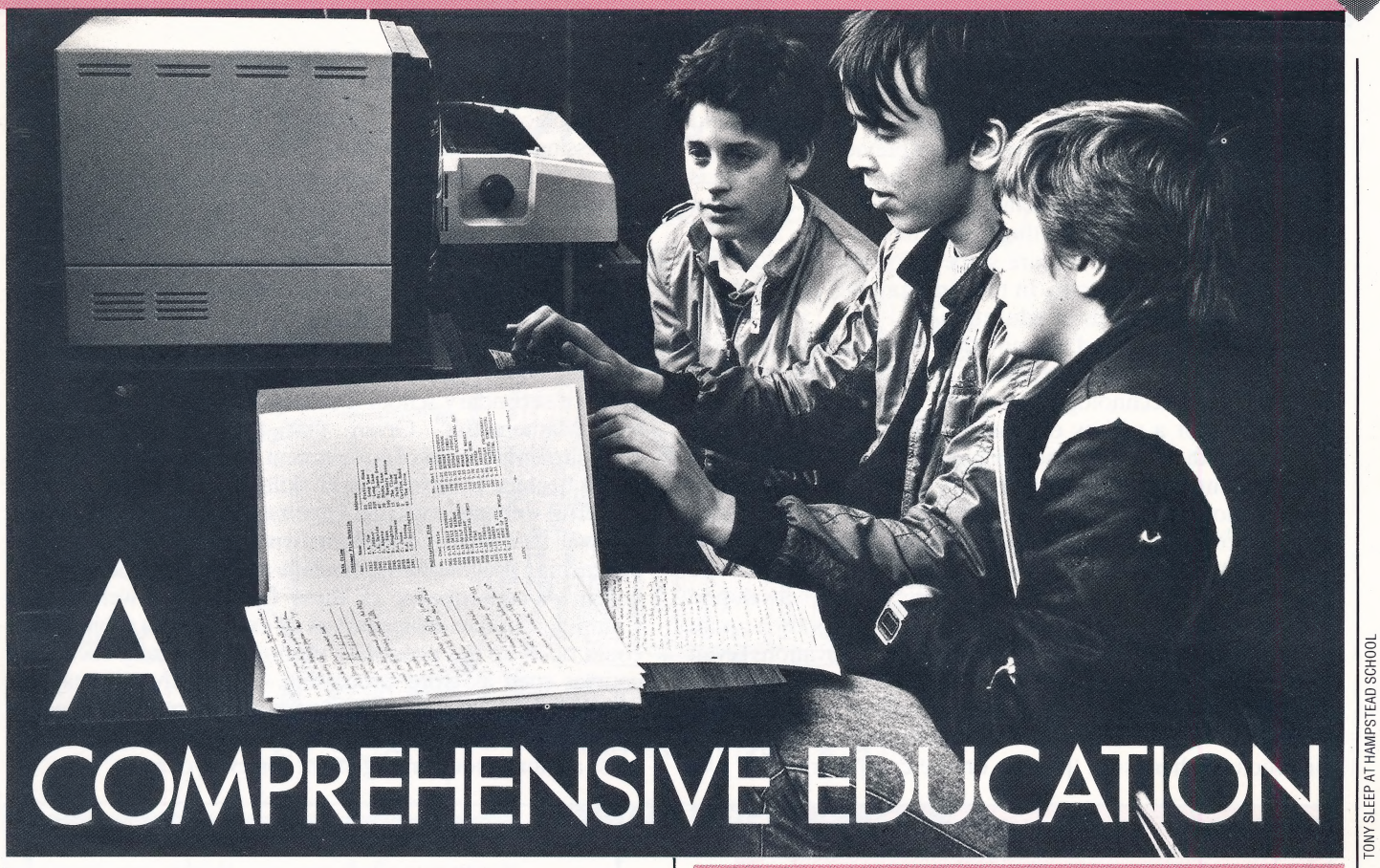
Coming Up

● A look at Inmos's transputer, and the advent of parallel processing.
● A review of the Commodore Amiga.

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editor** Jon Kaye; **Contributors** Mike Curtis, Steve Colwill, Steve Malone, Nigel Cross, David Fensome, Bob Page, Pete Connor, Peter Shaw; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd; Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; © **APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985; Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CANS1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited. Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY TONY SLEEP RETOUCHING BY HELEN ZAHORODNYJ / WISHFUL THINKING



A COMPREHENSIVE EDUCATION

TONY SLEEP AT HAMPSTEAD SCHOOL

The introduction of computers into the school syllabus in the 1970s and 1980s has been the subject of intense debate, both amongst teachers and parents. Here, we discuss the changing content of Computer Studies courses over the last decade, and show how present trends indicate that such courses have a limited future.

Early educational papers (in the late 1960s) were dismissive about any introduction of computers into schools. Not surprisingly, early examinations were very computer science-oriented, with little emphasis being placed on practical applications. Of course, the pupils then had no access to today's powerful micros and peripherals, which would have allowed them to create sophisticated 'real-life' systems.

During the 1970s, Computer Studies (as the subject came to be known) gained a lot of respectability simply by taking its place alongside the more traditional subjects as an O level and CSE examination, as well as continuing as Computer Science at GCE A level.

The syllabuses vary where computers are concerned throughout the country, but the more educationally 'enlightened' teachers will include such aims as 'to create an awareness of the impact of computers. . .' and 'to develop problem solving and communication skills. . .'. In the 1980s, the emphasis shifted away from teaching programming (especially BASIC) as a major part of the course, toward using existing packages to solve problems. Ironically, it could be argued that a knowledge of a programming language is perhaps the most vocational aspect of such a course, although this would obviously depend on the choice of language. In addition, the 'computer science' aspects of the subject (logic gates, half-adders and so on) mostly disappeared and the whole direction of the courses shifted towards the user and away from the technical aspects of the hardware (at least below A level).

Vocational Awareness
Despite the glamour of new technology, the study of computing itself is not growing as rapidly in popularity as one might think. Increasingly, pupils are abandoning the specific study of computers as they reach school-leaving age, preferring to combine an elementary awareness of computing with other skills, such as accounting, design and management

By 1985, the Examination Boards were consulting with schools in an attempt to create joint CSE and O level syllabuses for a new GCSE examination, as shown in the box. An essential part of the course involves problem-solving — the practical project required to be completed by each candidate. Not unexpectedly, the format of this project has altered considerably over the years, and tended towards a situation where the pupil is required to formulate and solve a problem using a computer. This solution can involve utilising existing packages, writing new programs or a combination of both. A typical project could be to create a simple system to maintain the files of videos and borrowers for a lending library.

A problem such as this is intended to have some counterpart in the real world and as such to have some meaning for the pupil. In practice, of course, the extent to which these aims are achieved will depend on the availability of hardware and software within the school. Even so, the project, as an exercise in real-life problem-solving, is (in educational terms) one of the strongest arguments for the inclusion of a course in computing in secondary schools.

A LEVEL COMPUTER SCIENCE

The typical GCE A level Computer Science syllabus extends the O level one in the depth of understanding required rather than in the content of the course. To some extent, however, it is more difficult to justify the examination at this level. The universities in general do not require A level Computer

Science as a prerequisite for a degree in computing, leading to the assumption that such a course starts from scratch. Similarly, many computer courses offered in Higher and Further Education Colleges assume no previous knowledge. Exactly where this leaves the status of the A level course is difficult to tell. One possibility is that tertiary education establishments could raise their expectations and, as in the case in other subjects, require some prior qualifications for Computer Science courses. In the long run, this would surely improve the final levels of education.

Outside the standard examination courses, many schools have introduced what are termed Computer Appreciation or Information Technology courses. These courses are introduced for a variety of reasons. In many cases, schools realise that there is a 'curricular gap' between the increasing use of computers by upper-primary school pupils and the examination courses in secondary schools, which do not commence until the fourth year. Others react to parental pressures to include courses on subjects that parents see as the 'knowledge of the future'. In addition, recently developed CPVE and 16-Plus courses for non-A level pupils also require an IT component. The extent to which these innovations (even where they are recognised) are tackled varies widely in practice. The *raison d'être* of such a course is often rather confused, with the result that the pupils end up with rather unstructured hands-on 'playing on the computer'.

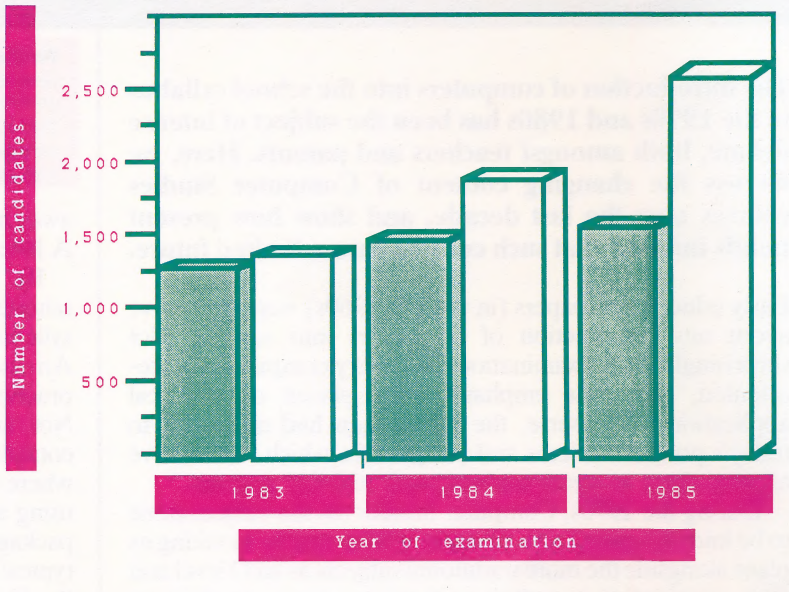
AIMS OF AN IT COURSE

Ideally, the aims of an IT course are to provide an insight into technologically based methods of collecting, storing, processing and disseminating information in all its guises (vocal, textual, pictorial, numerical, and so on). Focusing on methods where a computer can be utilised, the objectives are that all pupils should be confident in using a computer to collect, store and access information. In the process, they will use, for example, information retrieval packages, word processors or viewdata systems.

It is apparent that the required skills are relevant to all learning — many people argue that an IT course in the early years of secondary school is as essential as studying English and Mathematics. Given that there is a continuing commitment in terms of government funding, the age at which pupils become competent in IT skills is likely to fall. As more powerful software packages become available to schools, not only will the range of possible activities increase, but the stress on teaching programming, machine architecture and so on will lessen. A consequence of these factors may well be that Computer Studies disappears as an examination subject. This could open up the possibility of more specifically vocational courses such as word processing. Ultimately, we may see computers in schools as a 'service department', utilised by administration, teachers and pupils alike — quite possibly the most advantageous arrangement for everyone.

Specialised Study

These figures, released by the London Schools Examination Board, show the startling growth in the numbers of pupils taking O level Computing Studies. By contrast, however, the rise in the number of students sitting examinations at A level has been far less dramatic. This bears out the hypothesis that school leavers see a basic knowledge of computing as a valuable adjunct to other studies, but not necessarily desirable as a subject for specialisation



CAROLINE CLAYTON

Joining Hands

Sample joint examination syllabus (London/East Anglia):

Communication Skills

— Representation and interpretation of algorithms
— Need for and content of appropriate documentation

Basic Computer Science

— Hardware (CPU, memory, functional characteristics of input and output devices and backing storage)
— Computer architecture (registers, fetch-execute cycle)
— Machine representation (integers, characters, instructions)
— Software (high- and low-level languages, translators, errors and diagnostic aids, operating systems, application packages)

Information Processing

— Main aspects of systems analysis
— Data capture, validation, transmission and validation
— Methods of reducing errors (such as turnaround documents)
— File specification and design
— Specification of input and output documents
— Description of applications using systems flowcharts
— Methods of file recovery and security
— Methods of processing (batch, real-time, distributed, and so on)
— Limitations and suitability of computer use in given applications
— Social effects such as privacy and access to data

Problem-Solving Using A Computer

— The ability to describe a problem, capture the data, use a computer to process this data and present the results

FORMAL FUNCTION

The C programming language relies heavily on functions, which you can define yourself or which are provided by the system as standard. We examine the structure and role of functions in C, and discuss an example program that demonstrates relevant aspects of their use.

The C language is fundamentally based around the idea of a *function*. It is not a functional language as such, since the way in which each function operates is defined procedurally. However, virtually everything in C is defined in terms of the concept of a function.

What is meant by a 'function' in this context is a process to which are passed certain values — the arguments or parameters — and which uses those values to produce a single value that is returned. The values need not necessarily be basic types, like integers or reals, but may be structured types such as arrays.

There are several other essential concepts we need to consider — blocks, for example. In C, a *block* is a self-contained section of code, indicated by enclosure brackets ({}); in PASCAL, by way of comparison, a block is enclosed by begin . . . end statements. We also need to appreciate the idea of the *scope* of a variable — that is, the region over which it is available for use. Few versions of BASIC allow local variables, so many BASIC programmers will be unaccustomed to the idea of having to restrict the use of certain variables to certain portions of a program. In C, variables can be declared thoughout a program, and it is therefore particularly important to appreciate when and where they may be referred to.

VARIABLE DECLARATION

The basic rules of variable declaration in C are:

- When a variable is declared at the beginning of a function definition then its scope is the entire body of the function definition, including any other functions that may be defined within the original function body. In particular, since the program itself consists of the function main(), any variables declared at the beginning will have as scope the entire program — they are therefore *global* variables.

It is an increasingly important aspect of program design that such modules of a program should be completely self-contained and should use only locally declared variables and the formal parameters. If execution of a function involves data other than that in local variables, then this is known as a *side effect*. For example, operation of an input/output device is usually a side effect — as is the use of any global variable. Some side effects are beneficial — that is, they have no effect on the execution of other parts of the program — but others are harmful and their use should be avoided if at all possible. This particularly applies to the use of global variables.

- When a variable is declared inside a block, its scope extends over the remainder of that block only.

Variable declaration is particularly important since, as we will see later, it is part of the 'philosophy' behind C that a program should be constructed in modules. Each of these modules will exist on a separate file and the rules that govern how a variable may have scope to include functions defined in a different file are more complicated. We will look at these rules later.

There are two ways in which arguments or parameters can be passed to a function:

- *Value:* Where entirely new variables are created. These variables are local to the function and are initialised to the values passed by the calling routine.
- *Reference:* Where the address of the variable containing the parameter is passed. In this way, the same variable is used in the function and calling routine.

In C, all parameters are passed by value, but as we will see later, the language has extensive facilities for manipulating addresses or pointers to variables, and thus calling by reference can be obtained.

The formal syntax for a function definition is:

```
function_type function_name (formal_
   argument_list);
variable_declarations;
{
      body_of_function;
}
```

The function_type may be omitted if the value returned is int. The formal_argument_list is a list of variables of the appropriate types into which values will be placed when the function is used. The function is called by using:

```
function_name (actual_argument_list)
```

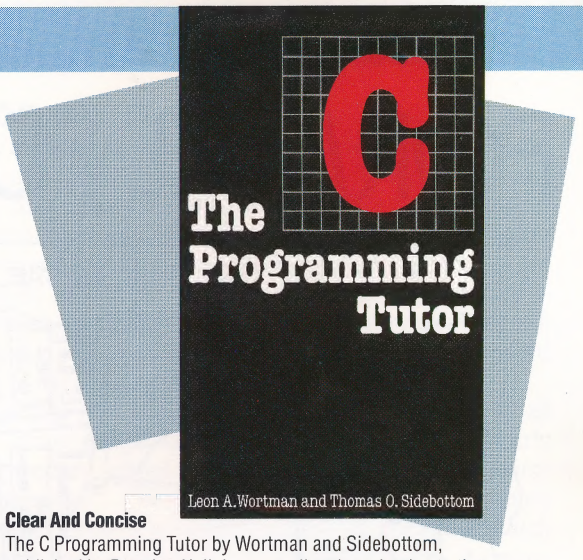
at any point where it is appropriate to use a value of the type that the function returns. However, if the type of the value returned is anything other than int, then the function name must have been declared as a variable of the appropriate type before it can be called. The actual argument list is the list of those values, or variables containing those values, which are to be passed to the function and placed in the formal arguments. The actual and formal arguments must, of course, agree in type and position in the two lists.

The final detail that we need to consider before writing a function is the use of the return statement to give the value that is to be returned to the calling routine. There may be any number of these statements, but one should be executed before exiting from the function.

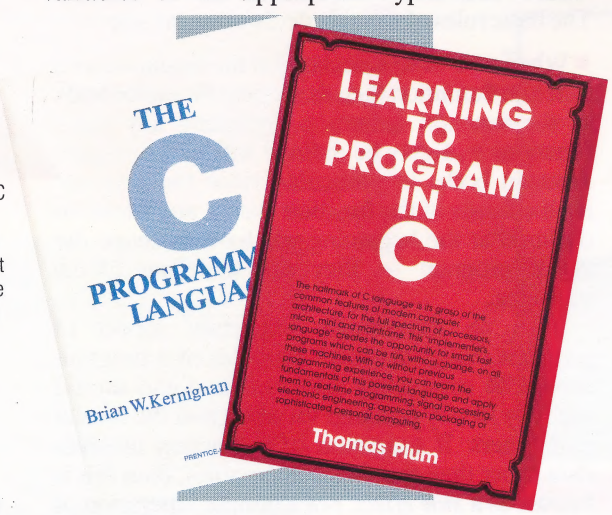
The following program will input a set of real numbers and print the largest and the smallest members of the set. The program uses three functions. One of these, printheading, has no arguments and no return statement, which means that the value returned by the function is not determined. As the returned value is not required by the calling routine, it is simply 'thrown away'. This function only has side effects, but they are all beneficial so we do not have to worry about them.

The program also uses the library function scanf, which performs formatted input in a very similar way to that used by printf for output. The first parameter to the function is a string that contains information on the format of the values to be entered, using the same technique as printf (see page 1725). The other arguments to the scanf function are the variables into which the values are to be placed. However, because parameters can only be passed by value, they cannot be used to return values to the calling routine. It is therefore necessary that these parameters be addresses of variables rather than the variables themselves. Each variable named in the call to scanf must be preceded by the address operator, &. We will be looking at other uses of this operator later.

```
#include <stdio.h>
main( )
{
   int count, size_of_input,
   double smallest, largest, number, min( ), max( );
/* note declaration of functions min and max */
/* note also that declaring inside the block makes
variables unavailable outside the block */
   printheading ( );
/* function called, value returned is assumed int and
is 'thrown away', i.e. not used anywhere */
   printf ("\nsize of input :");
   scanf ("%d",&size_of_input);
/* find out how many numbers are to be input */
   printf ("Now enter %d real numbers :\n", size_
```



Clear And Concise
The C Programming Tutor by Wortman and Sidebottom, published by Prentice-Hall, is an excellent introduction to the language for the beginner. Written in a clear and concise style, it introduces and explains complex subjects without confusing the reader and is particularly recommended to readers with no experience of programming languages other than BASIC. ISBN 0-13-110024-6, price £12.95



C Library
Publishers Prentice-Hall have been partly responsible for the increasing popularity of C by maintaining a comprehensive list of publications on the subject, including the seminal C Programming Language specification by Kernighan and Ritchie. A price of £22.95 might put this slim volume beyond the budget of many users, but until an international standard is approved it remains unsurpassed as a definitive specification of the language. Another relevant volume published by Prentice-Hall is Learning to Program in C by Thomas Plum, also shown here

```
    of_input);
/* get first number which will be the current largest
and smallest */
    scanf ("%lf", &number);
    largest = smallest = number ;
/* now get the other numbers */
    for (count = 2; count <= size_of_input;
    ++count)
    {
        scanf ("%lf", &number);
        smallest = min (smallest, number);
        largest = max (largest, number);
    }
    printf ("\nsmallest is %f\nlargest is
    %f\n", smallest, largest);
}
/* now come the function definitions */
printheading( )
/* no type needed so assumed to be int */
{
    printf("\n%s\n%s\n%s\n",
        "First enter the size of the set of numbers",
        "then enter that many real numbers.",
        "The largest and the smallest will be
        displayed");
}
double min(x,y);
double x,y;
/* note that the parameters are declared */
{
    if (x < y)
        return(x);
    else
        return(y);
}
double max(x,y);
double x,y;
{
    if (x > y)
        return(x);
    else
        return(y);
}
```

THE C PREPROCESSOR

Many compilers, for a variety of languages, feature *compiler directives* embedded in a program. These are certain lines that are recognised as instructions to the compiler rather than as statements in the language. C carries this idea further than most other languages and every C compiler incorporates a *preprocessor* that recognises certain *control lines* as instructions to alter the program accordingly before it is presented to the compiler itself. Control lines are distinguished by a hash (#) at the beginning. You will also notice that preprocessor directives do not need to be followed by semicolons, since they are not regarded as part of the program.

The #include <filename> or #include "filename" preprocessor directive instructs the preprocessor to include at this point the contents of the named file. There may be no difference between the use of angle brackets, < >, and quotation marks, " ", on some systems, but other systems may make a difference in the places which are searched in order to find the files. The usual convention is that the angle brackets are used for system files, while quotation marks are used for the user's own files. The functions printf and scanf that we have been using are included in a standard system file called stdio.h, so that all programs that use them should include the directive #include <stdio.h> — as at the beginning of our program.

It is possible to nest these calls so that one of the files that is included may also have a #include in it to incorporate another file.

The #define directive allows a certain limited form of macro substitution. In its simplest form, it allows the definition of a constant — for example, #define EOF -1 would cause the identifier EOF to be replaced by -1 at every occurrence throughout the program file. This feature makes it much easier to effect changes to 'constant' values if the need should arise. Other examples are:

```
#define PI 3.14159
#define EQ==
```

The generally accepted convention is that identifiers defined in this way are written in uppercase, reserving lowercase for the normal variables.

It is possible to include parameters in a macro definition, as in:

```
#define SUMSQ(x,y) ( (x * x) + (y * y) )
```

which gives an effect similar to the BASIC function definitions. An occurrence of, for example, SUMSQ(3,4) would be replaced by ((3 * 3) + (4 * 4)). Note the brackets included as a safety feature; the macro may be used in a situation where brackets are necessary, which means that it is always best to include them.

There is a #undef directive that causes the preprocessor to make no more substitutions for the named macro after it has been encountered.

The other most commonly used directive is the construct #if . . . #else . . . #endif, which allows conditional compilation. If the constant expression following the #if statement evaluates to true (non-zero) then the lines of code following the statement are compiled. If the constant expression evaluates to false (zero), then the lines following the #else are compiled. An example of its use is during the development of a program, when it may be helpful to include a lot of extra printf statements in order to keep track of what is happening. What we can do is put in directives of the form #define DEBUG 1, and everywhere we use an extra printf we put:

```
#if DEBUG
    printf (values . . .)
#endif
```

Then, when we have finished with the debugging, we need merely change the directives to #define DEBUG 0, and on recompiling our program will no longer include those statements.

SOLID GOLD

SAMNA WORD III

Wordwrap

★★★★★

Word III provides automatic wordwrap and full line justification.

Block Movement

★★★★★

All standard block commands are implemented, including column movement.

On-Screen Help

★★★★★

The program is one of the friendliest available in terms of the on-screen help provided.

80-Column Screen

★★★★★

The default screen sets the margins to 72 columns, but this can be quickly altered to use the full 80 columns.

Word Count

Not implemented.

Find/Replace

★★★★★

Up to 30 characters can be used in this facility, with a number of different search options.

WYSIWYG

★★★★

Although Word III contains some very useful facilities such as Zoom, other common ones (such as displaying on-screen underline) are missing.

Mailshot Facility

★★★★★

Comprehensive mailshot and database storage routines are provided by Word III.

Spelling Checker

★★★★

The Proof utility is fast and friendly although UK users may find themselves becoming impatient with the Americanised dictionary.

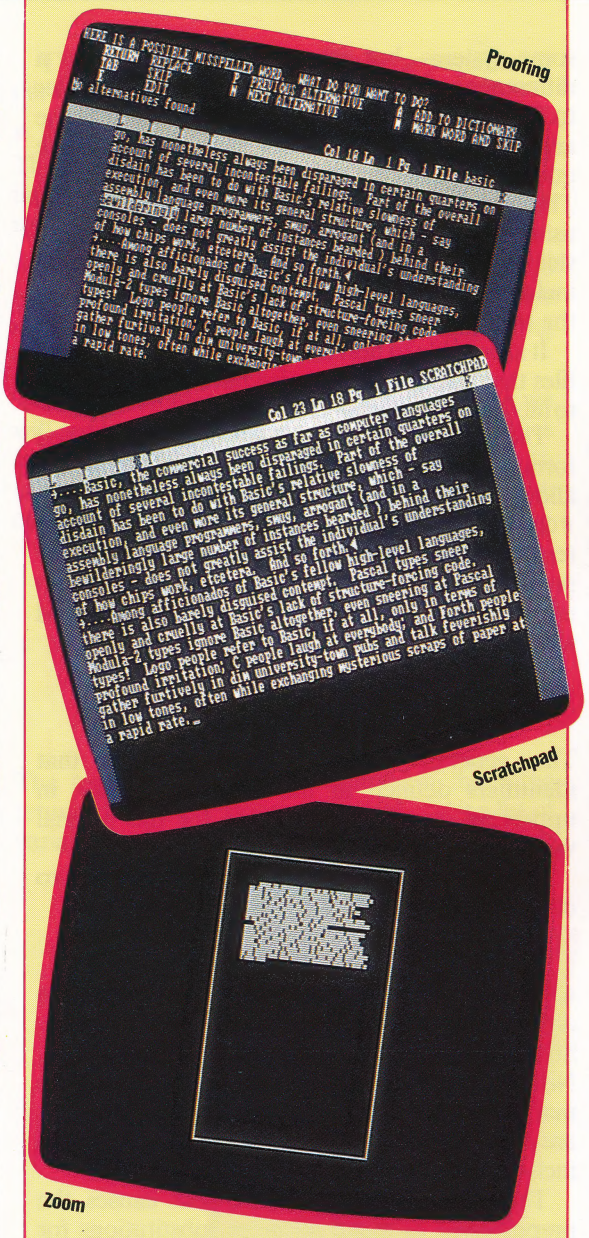
Founts Available

★★★

Although Word III supports subscripts and bold type, alternative founts are not available.

File Linking

Not implemented.



The Final Word

Samna Word III, although expensive, is a sophisticated word processing program that supports a number of facilities simply not available on cheaper systems. Apart from the usual insert and text manipulation functions provided on most programs, Word III has complex (yet user-friendly) spelling checking, mailshot and text formatting. Word III has a utility called Zoom, which gives the user some idea of how the final page will appear

Costing in the region of £550 and intended for the corporate business user, the Samna Word III package can be considered in the 'Rolls Royce' class of word processors. We take it for a test drive and see what luxurious appointments justify the price tag.

Like almost all the very expensive word processing packages, Samna Word III runs on the IBM PC range of computers, provided that they run under MS-DOS version 2.0 or above. This is because Word III implements file directories and pathways that were only introduced on DOS 2.0. The package consists of an introductory manual, layouts of the Samna keyboard for a number of different languages, a more detailed manual explaining each of the commands, and five floppy disks containing the program itself.

The disks contain a number of different utilities. The main program is held on two of the disks and other disks hold the printer routines and dictionary. The fifth, which is not actually part of the main program, is the training disk, which contains a number of short tutorial programs that guide you through the program.

Once the program is installed on a number of disks, the total program takes up over 700 Kbytes of disk space in 45 user files — a considerable amount for a single program. Running the program presents the user with a Scratchpad screen, indicating that no named file is currently loaded. On either side of the screen are coloured bars, indicating the margin positions, and above these another bar showing the default margin and tab settings. At the top of the screen are indicators showing the current cursor position, file name and a READY! prompt.

One of the greatest advantages of the IBM PC and its compatibles is the large number of programmable keys that have been made available to software developers. The programmers of Word III have taken full advantage of this and have not only utilised the 10 function keys to provide a series of word processing utilities, but have also reprogrammed some of the other keys for the same purpose.

The numeric keypad, for example, is used as a cursor cluster — a function normally used by the IBM keyboard — but some of the keys, such as 7, 9 and 1, have been used to move the cursor along a word, sentence and paragraph, respectively. On the other hand, the Scroll Lock key has been reprogrammed as the Mark key, which is used to define blocks that can be manipulated within the text. The function keys have been programmed to perform the most commonly used functions in word processing, such as underline and justify.

Word III, however, is much more comprehensive than the other programs we have looked at in this series. Calling a function, for instance, will often lead to another question asking the user exactly what area of the text that function is to be operated on — such as the whole document, the paragraph or simply everything after the current cursor position.

Samna's programmers have attempted to attain the often incompatible goals of maximum flexibility and ease of use. As an example, Word III has no automatic insert facility. When the user wishes to insert text in the middle of a document,

pressing the insert key will display an appropriate space. On Tasword II (see page 1735), this technique requires the user to realign the paragraph after inserting the text. On Word III, however, simply re-pressing the insert key frees the cursor from its position and automatically realigns the text, giving the best of both worlds.

Word III has an extremely useful help facility. The user can call it by pressing the Help key at the top left hand corner of the typewriter keyboard, which displays the range of options that are available. Alternatively, if the user hesitates before calling a function, the help screen will display itself automatically.

Several of the function keys are programmed to perform a wide range of miscellaneous operations. Among the most useful of these is the Do key. Among the utilities that can be entered from this are a number of the normal file functions, such as delete and copy, but the key is also used to access a number of operations not found on less expensive word processors.

THE ZOOM FUNCTION

When writing a large document, the screen is only able to display a window. However, should you wish to see how the whole document will look on a page, selecting the Zoom function will display a page layout with bars showing where the titles and paragraphs are positioned. This enables you to produce a more attractive layout without having to print it out a number of times.

Another function of the Do key overcomes another limitation of the window effect. When you are typing in a number of columns on a very wide document, for example, it may be necessary to refer to a column that is not displayed on screen. The Fold command will electronically 'fold' the document so that opposite sides of the page can be displayed on the same screen.

It's a requirement of business packages today to be compatible not only with the IBM PC, but also with other software, in particular with the widely used Lotus 1-2-3. In order to maintain compatibility with Lotus 1-2-3, Word III can translate ASCII files from disk and display them in Samna format. Lotus documents can thus be displayed and manipulated within Word III.

Any word processor that aspires to make an impact on the business market must have mailmerge and spelling checker facilities, both of which are implemented in Word III. The spelling checker, known as the Proof command, is one of the most comprehensive available on any package.

Like some utilities within the program, the spelling checker allows you to choose from a number of options, specifying what you want 'proofed'. Once this is decided, the computer then goes through each word, checking it against those held within its own dictionary. When it comes across a word it does not recognise, the computer will display a number of options, including a number of alternative spellings.

The mailshot utility implemented by Word III is known as Automatic Merge, and is a small database in which you can store a number of fields, such as first name, surname and address. These can then be fitted where appropriate into a standard document and printed. The program also allows 'wildcards' in searching for a particular record.

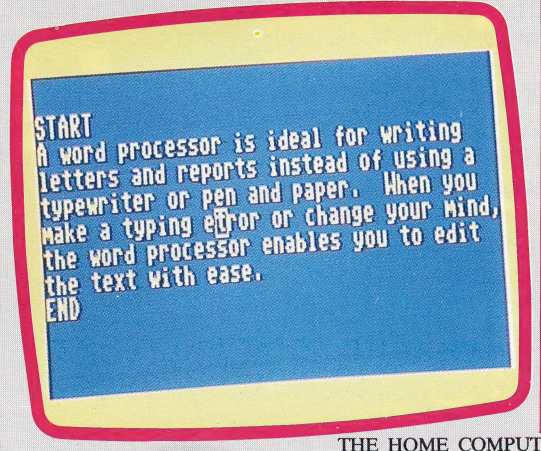
Samna's Word III is a very powerful word processing package that compares well with a number of dedicated word processing machines. However, the question remains: is the package worth the £550 price tag? If your business requires such a comprehensive word processing package, with facilities that are not available elsewhere, the answer is undoubtedly 'yes'.

The Small Print

At the opposite end of the scale from Samna Word III is Mini Office from Database Publications. The program, at around £6 for the cassette version, includes not only a word processing program but also database, graphics and spreadsheet utilities. It runs on the BBC Micro, Electron, Commodore 64 and Amstrad range of machines. Of course, the word processing package has very few of the sophisticated functions of Word III, and yet the program contains a number of features not found in more expensive programs.

Although the word processing facility in Mini Office is basic, many users who do not require complex print formatting functions may find that the program contains all they need to type letters and other documents. The program is menu driven, which allows the user to select a number of options, via the function keys, to manipulate the text. These include facilities such as loading, saving and printing documents.

The text screen itself supports automatic insert/delete and wordwrap, although the program does not allow justification. The top of the screen provides useful information — including a word count, the number of available characters remaining and a clock to tell you how long you have been typing. The program also allows text to be copied from one part of a document to another, although not via the normal block movement, but by simple copying of the characters from a previously defined cursor position to the current position



SCREEN SHOTS BY LIZ HEANEY

THERMAL PRINTER

A *thermal printer* forms characters on paper by applying the heating elements in its print head onto the page. Ordinary paper is coated with a heat-sensitive substance which, when melted, combines with another transparent substance to form a dye.

Thermal printers have become popular among home micro users due to their low cost and comparatively fast speeds. However, these printers do have their drawbacks — in particular, they are relatively expensive to operate, and there can be problems in obtaining the paper they use, which is generally provided only by the printer's manufacturer. Furthermore, many manufacturers use different chemicals that react at different temperatures. This means that one make of thermal paper will not necessarily work with a different machine. Nevertheless, thermal printers are fast, quiet and reliable.

Hot Stuff
Thermal printers are popular peripherals among home micro users, despite a few inherent drawbacks. Although reliable and relatively inexpensive, they can be very slow, produce low-quality print and require special paper

CHRIS STEVENS

THREADED LIST

A useful tool in the construction of databases and in artificial intelligence languages, a *threaded list* is a sequence of items to which extra links have been added, known as threads, allowing the computer to make additional connections between the data. Thus in a list, a series of threads could be added, which would, for example, extract every fifth item.

TIME SHARING

A technique that has become popular on computer systems, *time sharing* is a process whereby a number of users can have access to the same processor. Sharing can be performed either synchronously or asynchronously, as long as it is performed fast enough to create the illusion for each user of having sole access to the computer.

The primary advantage of time sharing is, of course, that it is much cheaper for a number of users to have access to a single machine. It is also more cost-effective because the computer will be working nearly all the time, instead of simply 'idling' when a sole user has finished with it for a while. The disadvantage of the system is that the computer has to be a high-performance machine capable of handling input from a number of different terminals and fast enough to make minimal any delays that might occur, requiring a high initial cost outlay.

TOP-DOWN DEVELOPMENT

Top-down development is the method of programming whereby the program is viewed as a whole and then broken down into smaller and smaller tasks. This is continued until the level is reached where the instructions that are to be implemented on the computer can be written.

Assuming we wanted to program a robot to go to the shop to buy milk, the top level would be 'go to shop, buy milk and return'. Each of these three actions can be broken down further. The instruction 'go to shop' can be divided, for example, into 'leave house, go down street, turn left, enter shop'. Then, in turn, we can break down the instruction 'leave house' into more detailed instructions, making the robot go to the front door, turn the handle and so on. By breaking a program down in this way, we eventually come to the level where we are instructing the computer to operate the robot's motors to move certain distances and perform very specific actions.

From this analogy, we can draw several conclusions. First, due to the way in which the program has been written, it will have a pyramidal structure, with tasks at higher levels each having subsets of smaller tasks. Thus each of the tasks can be a separate module that can be tested independently (see *modular programming*, page 1089). Secondly, there comes a point at the lowest level where the task cannot be broken down any further. Also, all divisions will have to depend on the imperatives set at the top of the program. Above all, top-down development ensures that programs are well structured, efficient, easily understood and simple to debug.

TRACE PROGRAM

A *trace program* is a piece of software that will follow the execution of a program and return reports on the status of it. As such, a trace program is a valuable tool in debugging. Although there are simple versions on a variety of home micros — usually activated by the command TRON (TRace ON) and deactivated by TROFF (TRace OFF) — a trace may offer a range of options to allow specific parts of the program to be followed.

For example, a trace program can either display each line of a program as it is executed or merely the names of the procedures that control is passed to. Equally, the programmer may wish to follow the values held by a particular variable. Used in this way, a trace program can pinpoint exactly where a program is at fault, and then identify and rectify the problem.

LIFTING THE LIDS

We round off our short series on computer hardware by taking the lid off three popular home micros to look at their individual architecture and identify in each machine the main components dealt with during this series. We also provide simplified logic diagrams to demonstrate how theoretical circuits are accomplished in practice.

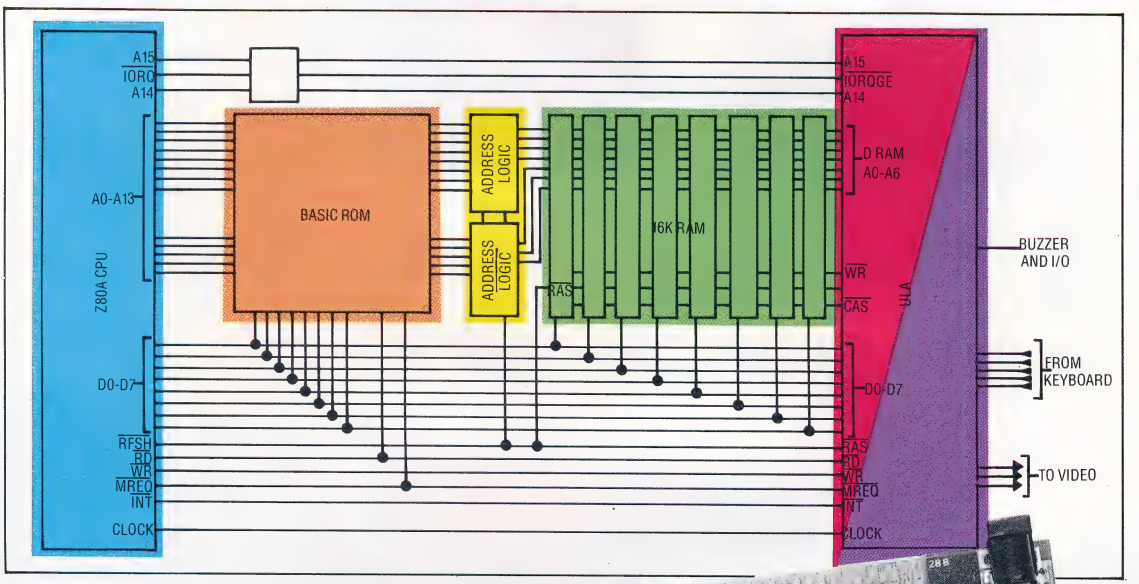
Sinclair Spectrum

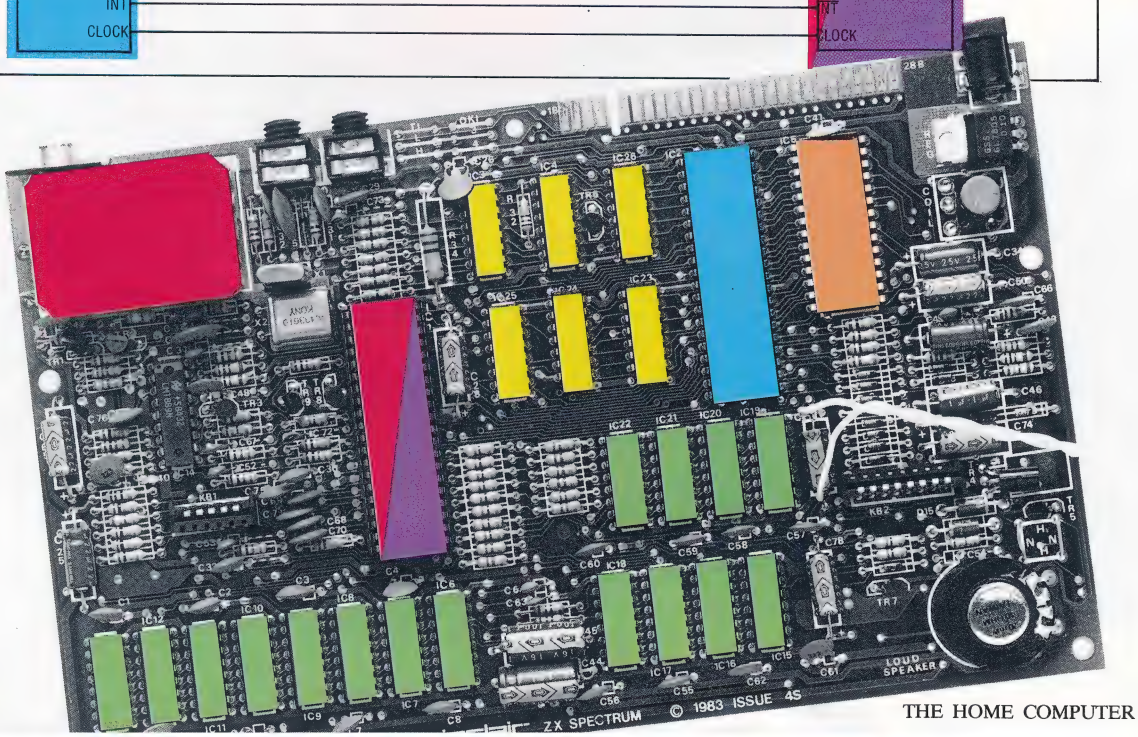
Sinclair's ZX Spectrum is rather different architecturally from the other machines we are looking at here. Based on the Z80A processor, most of the Spectrum's functions are performed by a special ULA. This custom-designed chip takes care of video processing, memory refresh and minimal I/O from keyboard, cassette recorder and buzzer. The circuit diagram shown here is for an early 16 Kbyte Spectrum. Later 48 Kbyte versions have extra RAM chips and address decoding.

As with all video display systems on micros, the ULA needs to make fast and regular accesses to memory. On the 48 Kbyte Spectrum, the first 16-Kbyte block, which contains the video data, is connected directly to the ULA. Under normal circumstances, the ULA can access this 16-Kbyte area independently while the CPU accesses the BASIC ROM or extra 32-Kbyte RAM area. The problem occurs when the CPU wants to access the 16-Kbyte video area (which it may well do as the BASIC system variables are stored here). To avoid the CPU and ULA coming into conflict, the ULA monitors address lines A14 and A15 and temporarily stops the CPU's clock if an access conflict occurs

Key

- Processor
- Dynamic RAM
- ROM
- PIO Subsystem
- Video Display Subsystem
- System Or Addressing Logic



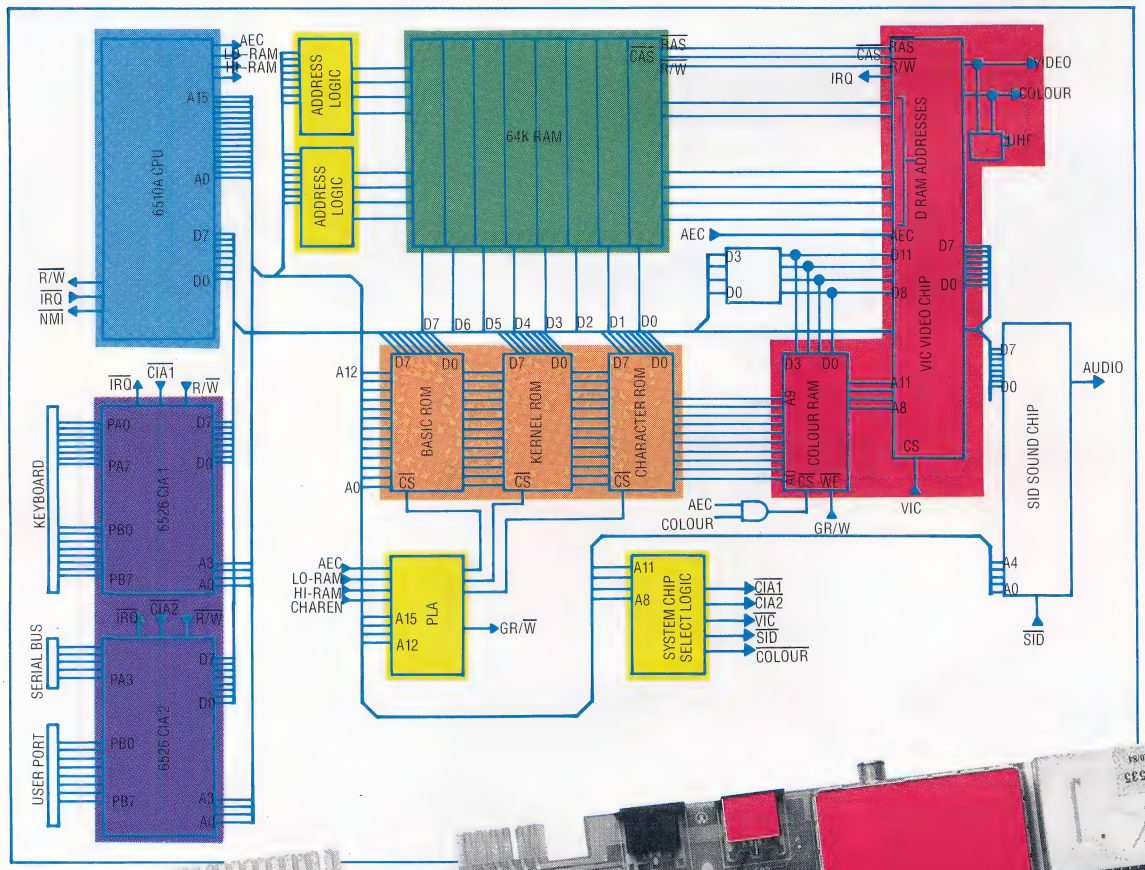


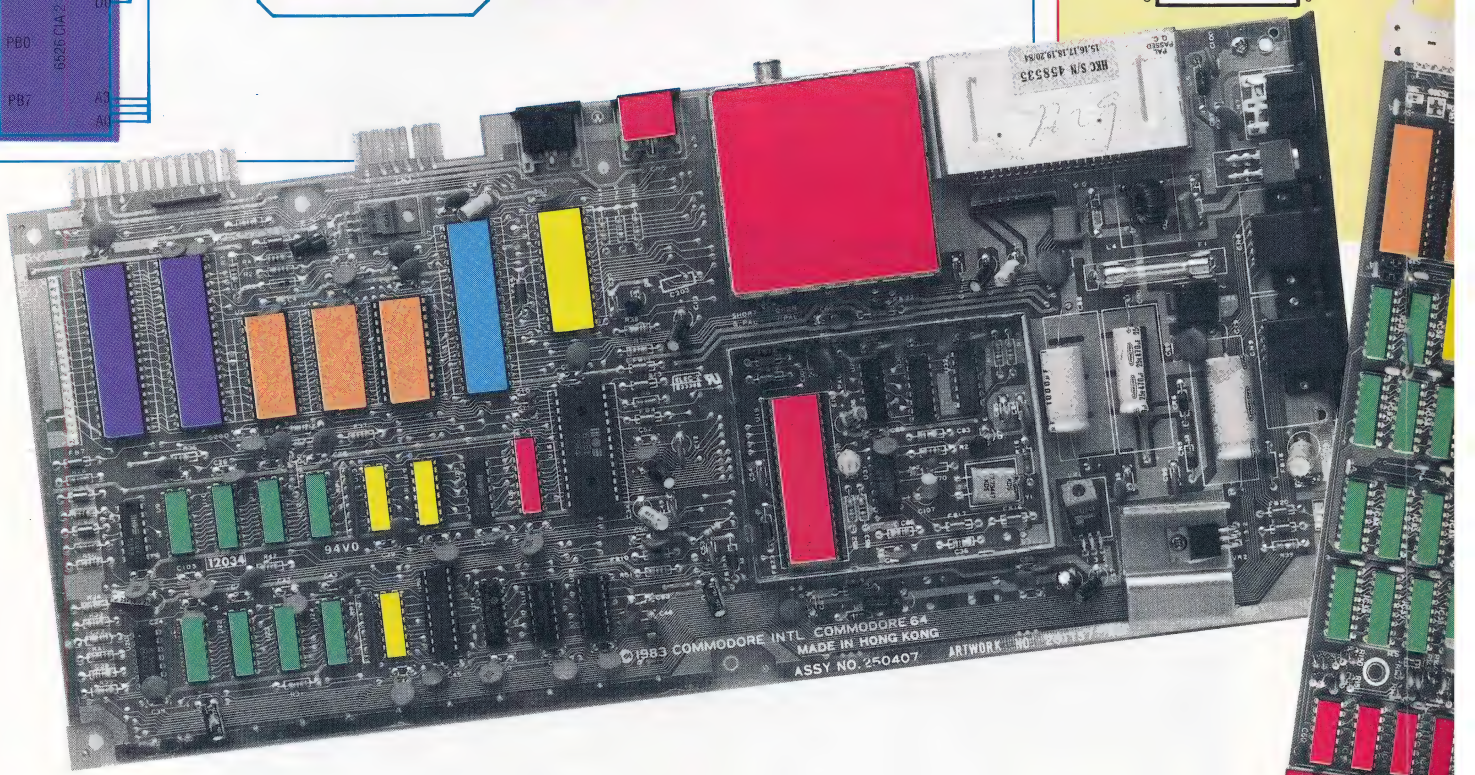
Commodore 64

From the hardware point of view the Commodore 64 is a well-equipped machine. It has 64 Kbytes of dynamic RAM, provided by eight 64-Kbit RAM chips, and three eight-Kbyte ROMs containing the BASIC, kernel I/O routines and character definitions. The 6510 processor is an enhancement of the 6502 that allows the ROMs to be banked in and out of the procesor's address space via an on-chip register and a special PLA.

The Commodore 64 has two PIO chips, one of which is dedicated to the keyboard. The second PIO uses one port to control the Commodore's own serial bus system, to which printers, disk drives and other peripheral devices can be attached; the other port is left vacant as a user port.

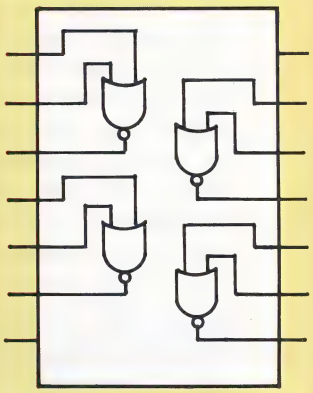
The VIC chip addresses the dynamic RAM directly to get screen information. As it needs more time communicating with memory than can be achieved by making accesses during the phase of the system clock when the CPU is not using the address and data buses, VIC temporarily halts the processor from time to time in order to address the RAM



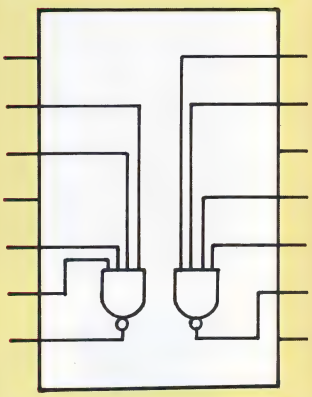


Logical Deductions

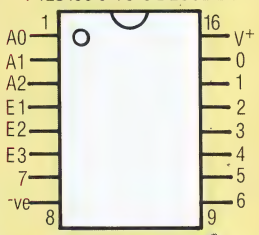
74LS00 QUAD 2-INPUT NAND



74LS20 DUAL 4-INPUT NAND

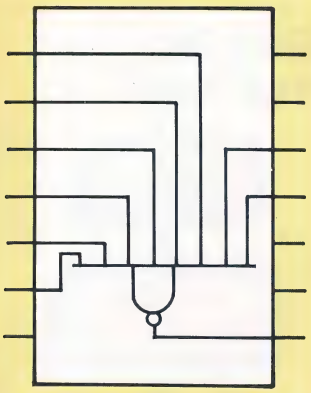


74LS138 3-TO-8 DECODER

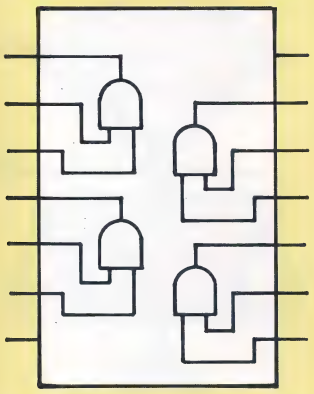


CHRIS STEVENS

74LS30 8-INPUT NAND



74LS02 QUAD 2-INPUT NOR



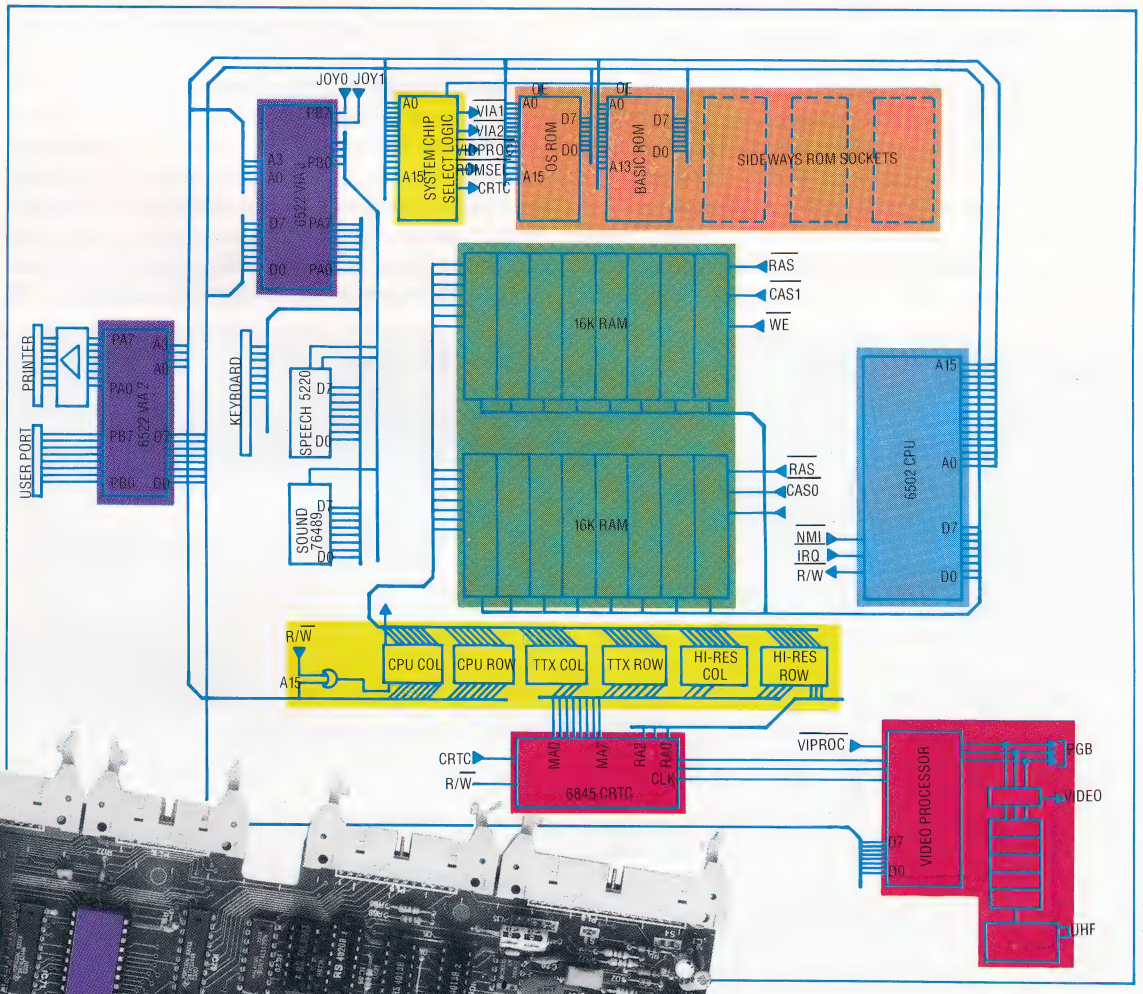
Applied Logic

Many of the smaller chips found on a micro PCB are from the 74** or 74LS** family. These chips contain various combinations of logic gates and are used for gating or decoding address or control signals. We show the details of several 74LS** logic chips here

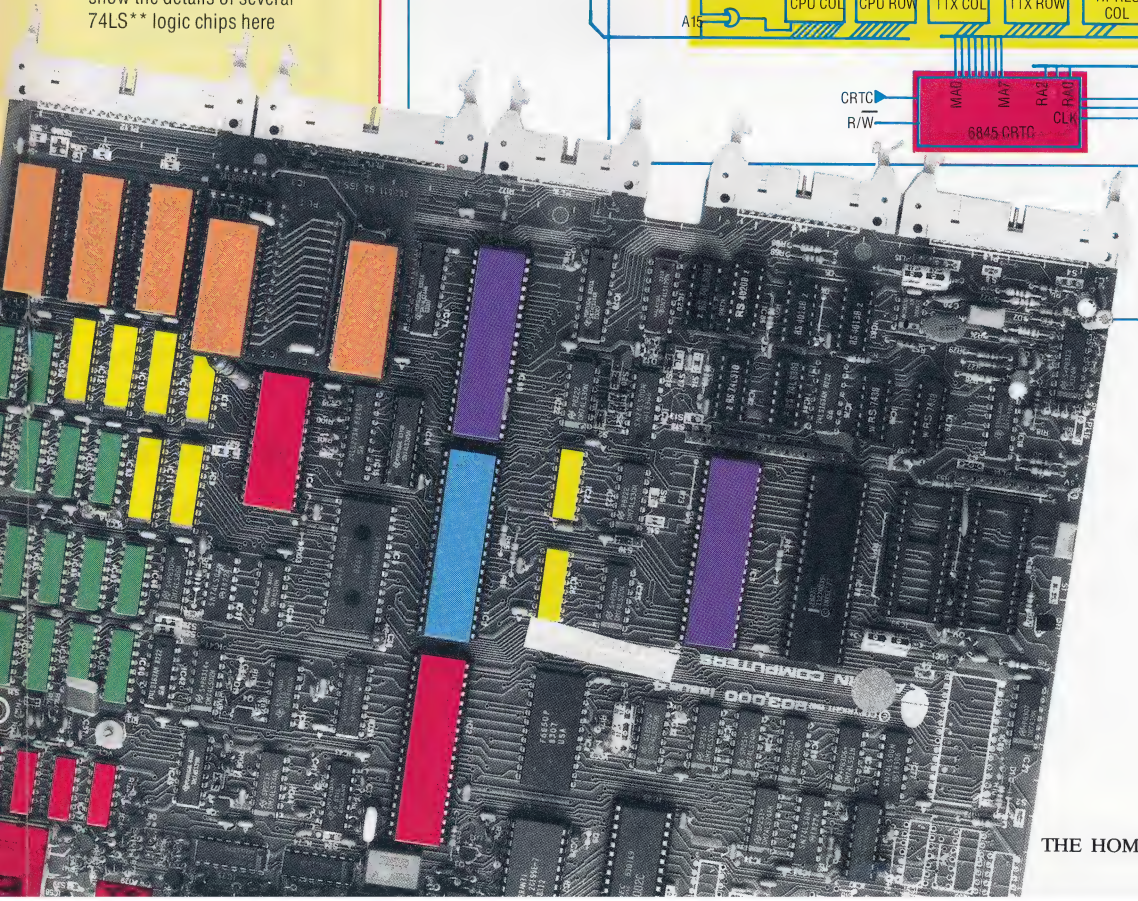
BBC Model B Micro

Although the BBC Micro provides most facilities that a user might want, the board shows its age. Originally designed in 1981, the BBC Model B still uses 16 16-Kbyte dynamic RAM chips to provide it with 32 Kbytes of memory. Contrast this with the Commodore 64, which requires only eight RAMs to provide 64 Kbytes. Six addressing logic chips are used to allow the CPU, video, and teletext chips to share the RAM. Two eight-Kbyte ROMs are provided in the basic configuration as well as three sideways ROM slots for extra ROMs.

Two PIOs are provided, as well as a serial I/O chip (not shown in the diagram). The first PIO is dedicated to the system and controls the keyboard, sound and speech chips. This PIO also monitors the joystick fire buttons and controls some of the video functions such as hardware scrolling. The CPU does not communicate with the keyboard and sound systems directly, but uses port A on the system PIO as, in effect, a slow data bus. The second PIO allows a parallel printer to be connected and the free PIO port remaining can be accessed via the user port



KEVIN JONES



LET'S DO THE TWIST

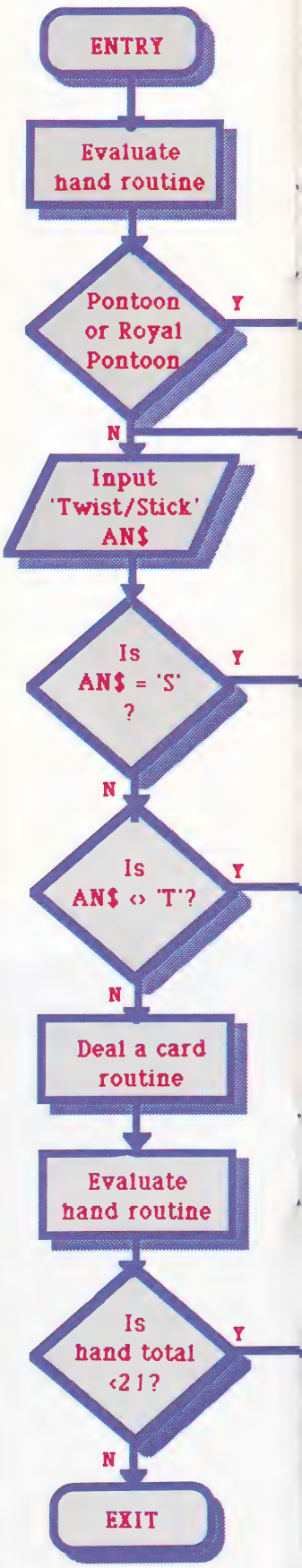
In the previous instalment of this series we looked at the routine that will evaluate a hand of cards in our pontoon game. Here, we develop the routines that complete the player's part of the game, analysing the hand in preparation for the computer's turn.

We've already developed a general-purpose routine that will total a hand and set a variable, EF, to various values corresponding to the possible states of the hand (see page 1746). We can now use this routine to allow the player's hand to be played out.

Remember that at this stage of the game, the player and the banker will have been dealt two cards each. The player can either stick or else twist to get close to 21 without busting. To tip the game a little further in the bank's favour, we have included the rule that prevents you from sticking if your hand totals less than 17. In a subsequent instalment, we'll include a third betting option, allowing you to double your bet and be dealt one more card.

Line 120 calls the twist/stick routine from the main game loop after the burn option has been checked for. The subroutine at line 2600 does not actually perform the work of twisting or sticking but instead calls another subroutine at line 2700 to do the work for it. On returning from this subroutine, you will have completed your hand and EF will be set to one of the five possible hand

Show Me Your Hand



The Player's Hand

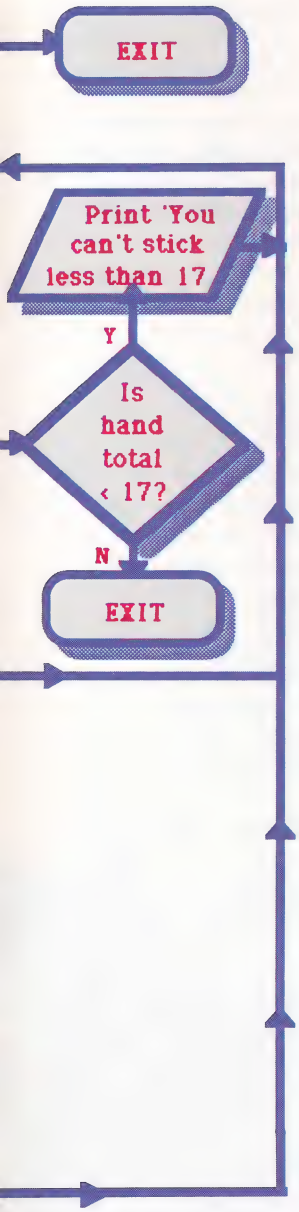
BBC Micro

```
 120 GOSUB 2600
 2600 REM
 2610 GOSUB 2700
 2620 ON EF GOSUB 3500,3600,3700,3800,39
00
 2630 RETURN
 2700 REM
 2710 GOSUB 800:IF EF=2 OR EF=3 THEN RET
URN
 2720 GOSUB 700:PRINT"TWIST/STICK/DOUBLE
";
 2725 AN$=GET$
 2727 IF AN$="S" THEN GOSUB 2800:IF CS=0
 THEN RETURN
 2730 IF AN$="S" AND CS=1 THEN 2700
 2750 IF AN$<>"T" THEN 2700
 2760 FL=0:PL=1:GOSUB 1300
 2770 GOSUB 800:IF EF=1 THEN 2700
 2780 RETURN
 2800 REM
 2810 CS=1:GOSUB 800
 2820 IF (TT(PL,2)>17 AND TT(PL,2)<22) O
R TT(PL,1)>=17 THEN CS=0:RETURN
 2825 GOSUB 700:PRINT"YOU CAN'T STICK UN
DER 17"
 2830 FOR DL=1 TO 5000:NEXT DL
 2840 RETURN
 3500 REM **** LESS THAN 21 ****
 3505 PV=1:PS=TT(1,2):IF PS>21 THEN PS=T
T(1,1)
 3510 GOSUB 700:PRINT"LESS THAN 21":FOR
DL=1 TO 500:NEXT DL:RETURN
 3600 REM **** ROYAL PONTOON ****
 3605 PV=4
 3610 GOSUB 700:PRINT"ROYAL PONTOON":RET
URN
 3700 REM
 3705 PV=2
 3710 GOSUB 700:PRINT"PONTOON":RETURN
 3800 REM
 3805 PV=0
 3810 GOSUB 700:PRINT"BUST":RETURN
 3900 REM
 3905 PV=3
 3910 GOSUB 700:PRINT"FIVE CARD TRICK":R
ETURN
```

Amstrad CPC Range

```
120 GOSUB 2600:REM tw
ist etc
2600 REM **** punter twist etc ****
2610 GOSUB 2700:REM do it!
2620 ON ef GOSUB 3500,3600,3700,3800,390
0
2630 RETURN
2700 REM **** twist/stick/double ****
2710 GOSUB 800:IF ef=2 OR ef=3 THEN RETU
RN:REM check pontoon/royal pontoon
2720 GOSUB 700:PRINT "Twist/Stick/Double
";
2725 an$="":WHILE an$="":an$=INKEY$:WEND
2727 IF an$<>CHR$(13) THEN PRINT an$
2730 IF an$="s" THEN GOSUB 2800:IF cs=0
THEN RETURN
2733 IF an$="s" AND cs=1 THEN 2700:REM c
an't stick
2750 IF an$<>"t" THEN 2700:REM input err
or
2760 fl=0:pl=1:GOSUB 1300:REM deal
2770 GOSUB 800:IF ef=1 THEN 2700:REM aga
in?
2780 RETURN
2800 REM **** stick ****
2810 cs=1:GOSUB 800:REM evaluate
2820 IF (tt(pl,2)>=17 AND tt(pl,2)<22)OR
 tt(pl,1)>=17 THEN cs=0:RETURN
2825 GOSUB 700:PEN red:PRINT "You can't
stick under 17"
2830 FOR dl=1 TO 1000:NEXT dl
2840 RETURN
3500 REM **** less than 21 ****
3505 pv=1:ps=tt(1,2):IF ps>21 THEN ps=tt
(1,1)
3510 GOSUB 700:PRINT "Less than 21":RETU
RN
3600 REM **** royal pontoon ****
3605 pv=4
3610 GOSUB 700:PRINT "Royal pontoon":RET
URN
3700 REM **** pontoon ****
3705 pv=2
3710 GOSUB 700:PRINT "Pontoon":RETURN
3800 REM **** bust ****
3805 pv=0
3810 GOSUB 700:PRINT "Bust":RETURN
3900 REM **** five card trick ****
3905 pv=3
3910 GOSUB 700:PRINT "Five card trick":R
ETURN
```

The flowchart shows how the player's part of the game is controlled. Exit from the controlling structure can be achieved under only three conditions: if the two cards originally dealt to the player give pontoon (an ace and a ten) or royal pontoon (an ace and a picture card); if the player sticks legally; or if the player busts



states, such as royal pontoon or bust. By using the ON. . .GOSUB command (with the exception of the Spectrum version, which takes advantage of the computed line numbers facility in Spectrum BASIC), the value of EF directs the program to a further subroutine. This prints a message informing you of the state of the completed hand, and sets a variable PV. This variable will be used later, when the computer takes its turn, to hold the state of your hand.

The actual twist/stick routine starts at line 2700. Immediately on entry, the evaluation routine is called. Both royal pontoon (an ace and a picture card) and two-card pontoon (an ace and a ten) are checked for in the two cards already dealt to the player. The routine continues by asking you to input T for twist or S for stick and then checks the keyboard for a response.

By using GET/INKEY$/GET$ instead of INPUT you can simply press the T or S keys without also having to press the Return key (pressing the Return key repeatedly can often get very annoying). The disadvantage of using GET/INKEY$/GET$ is that keypresses do not automatically generate the appropriate screen displays, as they do with INPUT. We therefore need to add extra lines to our program to print them.

If at this point you choose to stick, the program must check your hand to determine whether or not it is under 17. The routine at line 2800 tests this by looking at the hand scores for the player, which will have been calculated already by the hand evaluation routine. If your hand totals less than 17, a message is printed. In addition, a flag, CS, is set to 1. On return to the twist/stick routine, CS will determine whether or not you can legitimately stick, in which case the routine can be exited. If the program asks you to select again, however, it will loop back to the beginning of the routine.

If you select the twist option, you will be dealt a card and your hand will be re-evaluated. If your hand still totals less than 21 (signified by EF=1) after the new card is dealt, the routine loops back for a further input.

This completes the programming for the player's hand. In the next instalment we'll look at how the computer can be programmed to make an 'intelligent' response.

Sinclair Spectrum

```
120 GO SUB 2600: REM TWIST ETC
2600>REM **** PUNTER TWIST ETC EVALUATE
****
2610 GO SUB 2700: REM DO IT!
2620 GO SUB (EF*100)+3400
2630 RETURN
2700 REM **** TWIST/STICK/DOUBLE ****
2710 GO SUB 800: IF EF=2 OR EF=3 THEN  R
ETURN : REM CHECK PONTOON/ROYAL PONTOON
2720 GO SUB 700: PRINT "TWIST/STICK/DOUB
LE ";
2725 LET A$=INKEY$: IF A$="" THEN  GO TO
 2725
2727 IF A$<>CHR$ 13 THEN  PRINT A$
2730 IF A$="S" THEN  GO SUB 2800: IF CS=
0 THEN  RETURN
2733 IF A$="S" AND CS=1 THEN  GO TO 2700
: REM CAN'T STICK
2750 IF A$<>"T" THEN  GO TO 2700: REM IN
PUT ERROR
2760 LET FL=0: LET PL=1: GO SUB 1300: RE
M DEAL
2770 GO SUB 800: IF EF=1 THEN  GO TO 270
0: REM AGAIN?
2780 RETURN
2800 REM **** STICK ****
2810 LET CS=1: GO SUB 800: REM EVALUATE
2820 IF (T(PL,2)>=17 AND T(PL,2)<22) OR
T(PL,1)>=17 THEN  LET CS=0: RETURN
2825 GO SUB 700: PRINT  FLASH 1;"YOU CAN
'T STICK UNDER 17!"
2830 FOR L=1 TO 300: NEXT L
2840 RETURN
3500 REM LESS THAN 21
3505 LET PV=1: LET PS=T(1,2): IF PS>21 T
HEN  LET PS=T(1,1)
3510 GO SUB 700: PRINT "LESS THAN 21": R
ETURN
3600 REM **** ROYAL PONTOON ****
3605 LET PV=4
3610 GO SUB 700: PRINT "ROYAL PONTOON":
RETURN
3700 REM **** PONTOON ****
3705 LET PV=2
3710 GO SUB 700: PRINT "PONTOON": RETURN

3800 REM **** BUST ****
3805 LET PV=0
3810 GO SUB 700: PRINT "BUST": RETURN
3900 REM **** FIVE CARD TRICK ****
3905 LET PV=3
3910 GO SUB 700: PRINT "FIVE CARD TRICK"
: RETURN
```

Commodore 64

```
120 GOSUB 2600:REM TWIST ETC
2600 REM **** PUNTER TWIST ETC EVALUATE
****
2610 GOSUB2700:REM DO IT!
2620 ON EF GOSUB3500,3600,3700,3800,3900

2630 RETURN
2700 REM **** TWIST/STICK/DOUBLE ****
2710 GOSUB800:IF EF=2 OR EF=3 THEN RETUR
N:REM CHECK PONTOON/ROYALPONTOON
2720 GOSUB700:PRINT"TWIST/STICK/DOUBLE "
;
2725 GET AN$:IF AN$="" THEN 2725
2727 IF AN$<>CHR$(13) THEN PRINT AN$
2730 IF AN$="S" THEN GOSUB 2800:IF CS=0
THEN RETURN
2733 IF AN$="S"AND CS=1 THEN 2700:REM CA
N'T STICK
2750 IF AN$<>"T" THEN 2700:REM INPUT ERR
OR
2760 FL=0:PL=1:GOSUB1300:REM DEAL
2770 GOSUB800:IF EF=1 THEN 2700:REM AGAI
N?
2780 RETURN
2800 REM **** STICK ****
2810 CS=1:GOSUB800:REM EVALUATE
2820 IF (TT(PL,2)>=17 AND TT(PL,2)<22)OR
 TT(PL,1)>=17 THEN CS=0:RETURN
2825 GOSUB700:PRINTCHR$(28);"YOU CAN'T S
TICK UNDER 17"
2830 FOR DL=1 TO 1000:NEXT DL
2840 RETURN
3500 REM **** LESS THAN 21 ****
3505 PV=1:PS=TT(1,2):IF PS>21THEN PS=TT(
1,1)
3510 GOSUB700:PRINT"LESS THAN 21":RETURN

3600 REM **** ROYAL PONTOON ****
3605 PV=4
3610 GOSUB700:PRINT"ROYAL PONTOON":RETUR
N
3700 REM ****  PONTOON ****
3705 PV=2
3710 GOSUB700:PRINT"PONTOON":RETURN
3800 REM ****  BUST ****
3805 PV=0
3810 GOSUB700:PRINT"BUST":RETURN
3900 REM ****  FIVE CARD TRICK ****
3905 PV=3
3910 GOSUB700:PRINT"FIVE CARD TRICK":RET
URN
```

CAROLINE CLAYTON

DIRECTORY ENQUIRIES

From searching a directory to printing out a file, the resident commands of the 16-bit operating system MS-DOS take care of most eventualities. Before examining each one in detail, let's first boot up the system and see what options are initially open to us.

MS-DOS systems are normally booted just by powering up (on a hard disk system) or inserting a floppy system disk. Many machines have IBM-style keyboards, and a reset, or 'warm' boot, can be effected on these at any other time by the traditional IBM PC 'Control-Alt-Delete' method — two keys on the left-hand side of the keyboard, labelled CTRL and ALT, are held down while simultaneously depressing the DEL (delete) key on the right-hand side. The last operation, after MS-DOS has loaded and performed the system initialisation, is to look on the default drive for a 'batch' file named AUTOEXEC.BAT. If it is present, the commands that it contains will all be executed automatically as if they had been typed in from the keyboard.

We could, for example, write a simple AUTOEXEC.BAT file containing the command:

```
MENU
```

where MENU.COM (or MENU.EXE) was an appropriate application program (EXEcutable or COMmand file). This program would then be executed automatically every time the system is booted up with this particular disk — this is known as a 'turnkey' system.

MS-DOS (and PC-DOS) has great power and flexibility, particularly versions 2 and 3, and all the 'transient' utilities are provided as separate EXE or COM files. In this instalment of our MS-DOS series, however, we'll concentrate on the nucleus of 'resident' commands common to all versions of MS-DOS and PC-DOS. These are available via the DOS command line interpreter, or 'shell', called COMMAND.COM (the only part of DOS that is 'visible' to the user).

Perhaps the most frequently used of all systems utilities is the directory command. This lists all the files on a disk or, from DOS 2 onwards, on a 'directory' (a named and restricted portion of the whole disk space). The basic form of the command is just:

```
dir
```

As with CP/M, on which DOS 1 was closely modelled, this simple form lists all files on the currently logged ('default') disk. Should a directory listing of another drive be required, this can be specified as a command line parameter. Again, this is identical to the CP/M usage and the same syntax for ambiguous file specifications is also adhered to — ? represents any single character, and * may be used for either a group of characters or, possibly, no characters at all.

From one to eight characters are allowed in the primary file name, and up to three characters (or none) for the extension or secondary name — again, just like CP/M. When the listing appears on the screen, however, you will notice a radical difference both in the format used and the amount of information displayed. MS-DOS lists not only the file names, but also the file size (in bytes) and the date and time that each file was written to disk.

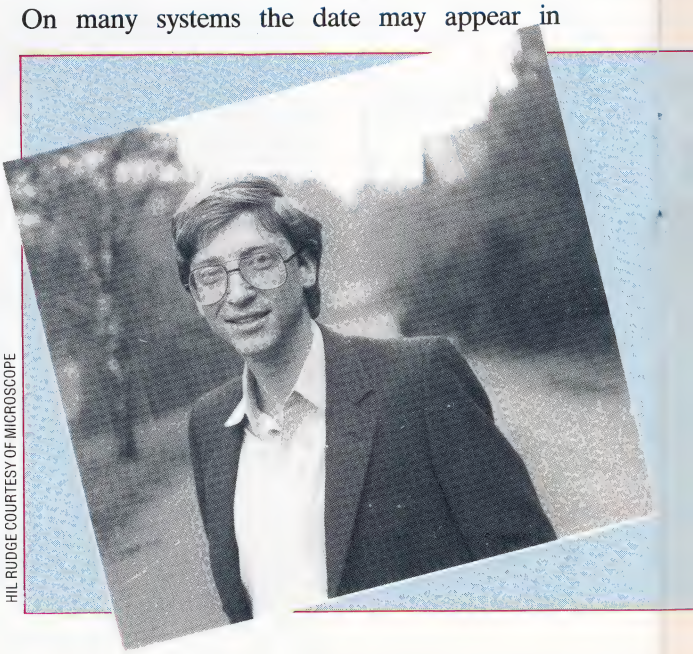
Here is a typical directory listing:

```
A>dir

Volume in drive A has no label
Directory of A:

COMMAND COM   22672   9-03-85  11:36a
AUTOEXEC BAT    128  22-07-85   5:15p
WP       EXE  73156  12-10-84  12:07p
WPMSG    OVR  38269  27-09-84  12:02p
WPINST   EXE  56385   3-10-84  12:04p
WCOUNT   PAS   4986   1-08-85   3:11p
WCOUNT   OBJ   3874   1-08-85   3:12p
WCOUNT   EXE   8385   1-08-85   3:14p
WCOUNT   BAK   4732  31-07-85  11:23p
MARY1    TXT    634   7-08-85  11:05a
INVOICE1 885    477  21-08-85   9:36a
MARY2    TXT    938  15-08-85  12:47p
```

On many systems the date may appear in



HIL RUDGE COURTESY OF MICROSCOPE

American format (Month-Day-Year), but recent versions of DOS cater for the UK convention as well as allowing special character sets for European and Scandinavian languages. Although sizes are given in bytes, files will occupy more space than this on the disk. Every file will use a certain whole number of sectors — typically 512 bytes each.

FILE STATISTICS

DOS also displays the number of files and the remaining space on the disk. This combines many of the features of the external ('transient') CP/M utility STAT.COM with that of the resident dir command and, together with the file creation time and date stamp, makes for a much more useful command. The main reason for the added functionality is the removal of the 64 Kbyte memory barrier.

CP/M systems were almost invariably restricted to this maximum address space unless sophisticated memory 'paging' was implemented. Every few bytes used by the operating system robbed the user of that much free RAM, so that only the essentials were built into the system code as resident commands. Detailed information (such as file size and attributes) was obtained by executing a separate program (like STAT) as a transient utility.

MS-DOS also has a wide and versatile range of these programs, but because of the greatly increased memory space on 16-bit systems (256 Kbytes and upwards is normal), the resident system can be much larger and consequently contain much more powerful features. Version 3 of DOS may easily require over 60 Kbytes of RAM — exactly how much depends partly on the OEM (original equipment manufacturer) supplying the system. A system clock is assumed by MS-DOS and, even if there is no actual hardware device (with battery back-up) to keep ticking over when the computer is switched off, a software emulation will be available, usually working on a continual interrupt method.

MS-DOS Resident Commands

Command		Function	Switches	Usage	
	copy	copy files	/a /b /v	copy	{drive:}wildname {target}
11 10 85	date	display /set date		date date	{dd-mm-yy} {mm/dd/yy} (US version)
	del	delete files		del	{drive:} {wildname}
	dir	list files	/p /w	dir	{drive:} {wildname}
	erase	see del			
	ren	rename files		ren	{drive:}name1 name2
	rename	see ren			
12 20 52	time	display /set time		time	{hh:mm:ss}
	type	list text files		type	{drive:}filename

Key:

{ } items appearing in braces are optional

| a vertical bar separates alternatives

character ::= any ASCII character except " , . ? * < > ; : [ ] / \ |
primaryname ::= character{character} (eight maximum)
secondaryname ::= {character} (three maximum)
filename ::= primaryname{.secondaryname}
wildname ::= filename containing 'wild card' characters (* and ?)
device ::= CON | LST | PRN |AUX | LPT1 etc
target ::= filename | device
drive ::= A | B | etc. (typically up to P)

ICONS: CAROLINE CLAYTON

In this case, the software clock will be initialised to, say, 1-01-85 12:00:00 and must be set manually when the system is booted by means of the DOS commands date and time. These display the current date (or time) and invite entry of new values from the keyboard, which must be in the same formats given earlier, though not necessarily complete. The interaction can also be curtailed by entering new values on the command line. So, for example:

```
time 14
```

would set the system clock to 14:00:00 (the last two digits represent hundredths of a second, though the resolution of time is not often as high as this). Any file created a little later will be stamped with the current system values and might appear in the directory listing as, say:

```
NEWFILE    TXT     512  21-08-85   2:05p
```

All MS-DOS programs are given the extension EXE (executable) or COM (command), the only difference being in the restrictions as to the memory locations into which they can be relocated and run. Generally speaking, EXE files are more flexible and can be loaded at a 'paragraph' (computer word) boundary, whereas

O Lucky Man

The story of how Bill Gates of Microsoft secured the contract to supply IBM with a 16-bit operating system (MS-DOS) is a little unusual. Having been approached by IBM, Microsoft was initially uncertain whether it could accept the contract to produce the OS. Consequently, IBM decided to pay a visit to Digital Research to discuss the matter with Gary Kildall, who was renowned for writing CP/M.

On the fateful day of the visit, however, Kildall was out of the office on a flying trip. After waiting two hours for him, the IBM delegation departed and the contract was subsequently awarded to Microsoft. In this way, a contract that eventually led to the creation of an industry standard was awarded to a company known only for a version of BASIC and with little experience in operating system technology

COM files must reside at a 'segment' boundary (at the start of a 64 Kbyte memory area or segment, as with the Intel 8086 family conventions). Other secondary file names are used with special meanings, sometimes merely for convenience, so that documents, letters and any other text files are often given the extension .TXT (or .DOC) as an *aide-memoire*. Machine code object files in standard Intel (relocatable) format will have the extension .OBJ, the source code being named with the usual .PAS, .FOR or .C extensions for PASCAL, FORTRAN and C programs.

Studying the directory listing further, we can deduce that the program WCOUNT.EXE originated as a PASCAL source file (WCOUNT.PAS saved at 11 minutes past three on the afternoon of the 1st of August). This was compiled to a standard Intel format .OBJ file about one minute later, and just two minutes later was linked to produce the final EXEcutable program. There is still evidence of the previous version (WCOUNT.BAK) archived as a back-up (BAK) file by the word processor (WP.EXE), which obviously gave some trouble as the programmer appears to have been working quite late at night the previous week (on the 31st of July). Perhaps, judging by its name, we could hazard a guess that this program counts words in a text-file — all this information from just a quick study of the information from dir!

The command has a couple of tricks up its sleeve that further distinguish it from its CP/M equivalent. These take the form of 'switches', or options appended on the command line. The P switch gives a page (or screenful) of files at a time, pausing until a key is pressed before continuing the listing. This is particularly useful on modern high-capacity disks when hundreds of files could be stored on one device or directory. The other switch, W, requests suppression of the detailed information, and causes only the filenames to be listed in Wide format — five to a line. Many DOS commands, internal and external, have such switches and they are signified as such by preceding *each* of them with a forward slash:

```
dir c:*.exe/p/w
```

This would cope with up to 115 (5 × 23) .EXE files per screen 'page'.

THE TYPE COMMAND

Another command that behaves like its CP/M equivalent is the one most often used to display the contents of a text file on the screen (or printer). The syntax is:

```
type{device:}filename
```

So, for example, to list a MODULA-2 program called MOUSE.MOD on disk device B, we would enter:

```
type b:mouse.mod
```

As with CP/M systems, issuing a Control-P before the carriage return will activate the printer, should we want a hard copy listing.

There is an alternative method to CP/M's, which is in some ways preferable, and certainly more flexible. Whereas CP/M uses PIP.COM as a transient system program for data transfer, MS-DOS has a very powerful COPY utility built in as a resident command. In its simplest guise it is invoked with:

```
copy{device:}sourcename{target}
```

where target can be another filename or a system device such as CON (the console), LST or PRN (the standard list device or printer), respectively. So, to get a printout:

```
copy b:mouse.mod prn
```

will achieve the same as the type command above, but without the necessity to use Control-P while avoiding anything other than the contents of the file from appearing on the listing. With the type command, a system prompt (such as A>) would be appended.

The copy command can take three optional switches: /a copies ASCII files using an end of file marker (Control-Z), /b uses the physical end of file, ignoring any Control-Zs, and /v verifies each data transfer (permanently switchable with 'verify' on). This is possible because MS-DOS keeps a record of how long each file is in the directory. If the target is not specified, copy assumes you mean a file of the same name as the source(s), but on the default drive (such as copy c:*.txt/a/v).

There are many other possibilities for this powerful resident command. Among the more interesting of these is the ability to concatenate files. If we say:

```
copy b:*.txt bigdoc.txt
```

all text files on device B will be accumulated in the file bigdoc.txt on the default drive (in the directory order). If we want to control the ordering or specify files with different extensions, the + sign can be used:

```
copy c:x1.wrk+x2.dat+b:x3.frm xxx.new
```

Appending to an existing file is achieved by omitting the target:

```
copy page1.txt+page2.txt+page3.txt
```

with the contents of page2.txt and then page3.txt being added on to the end of page1.txt, preserving this name for the whole file.

There is a system editor (called EDLIN.EXE), and we will take a look at this and some of the other transient 'commands' in the next instalment. In the meantime, a text file can be created without the aid of any form of word processor by simply giving the command:

```
copy con newfile.txt
```

After entering any desired text, the file is closed and written to disk by entering Control-Z (the end of file marker). Unlike pip (CP/M), this must be sent by following it with ENTER <CR>.

FOLLOWING INSTRUCTIONS

We continue our examination of the Motorola 68000's instruction set. Having looked, in the previous instalment, at the data copying and simple computational instructions, we now move on to discuss more complicated arithmetical operands. First, however, we need to consider the use of the 68000's 'compare' instruction.

Closely related to the SUB instruction (see page 1758), the CMP or compare instruction is extremely important because no matter what programming application area we work in, there will always be decision elements in our program design. So, consider the following high-level design, which includes a decision pseudo-code element:

```
If inputchar='N' then
      cleararray
end
```

Here, the If decision will be coded in the implementation phase of the project as:

```
CMP.B  #'N',D0     Compare the input byte
                   with N
BNE    NOTSAME     Goto NOTSAME if
                   different
```

When the CMP instruction is executed, the source is subtracted from the destination and only the condition codes are changed — the destination is not affected (as, for example, in the SUB instruction). The conditional branch, BNE, which follows will cause a branch to the label NOTSAME if the result of the subtraction of D0—'N' is not equal to zero.

Let's look at two more examples, which use different data attributes:

```
CMP.W SPEED,D3     Compare the word contents of
                   location SPEED with D3
CMP.L  D1,D2       Compare the full long words in
                   D1 and D2
```

Notice that the destination fields in these examples are data registers, and that in fact any addressing mode can be used for the source addressing mode. If you require to address any other form of destination then different CMP instructions are used. For example:

```
CMPA   BETTY,A3    Compare the contents of
                   location BETTY with A3
```

In fact, any source addressing mode can be used. However, the immediate form that follows only data-alterable modes are allowed.

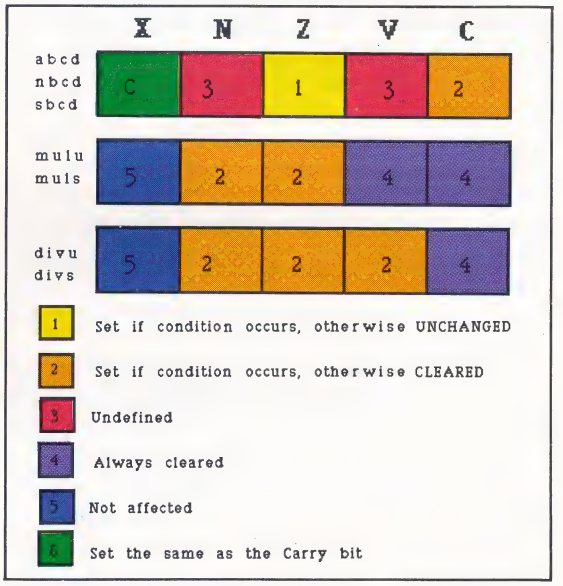
```
CMPI   #3,(A4)     Compare whatever A4 is
                   pointing to with 3
```

One final CMP variation worthy of special consideration is the CMPM instruction. For example, CMPM (A2)+,(A3)2 will compare the contents of the memory locations pointed to by A2 and A3 and then post-increment the pointers. So this one instruction may be used, say, to compare stored keywords with characters in an input buffer — and all in one word too! Here is an example:

```
LEA    KEYS,A2     Set up first pointer to the
                   keywords
LEA    BUFFER,A3   And the buffer pointer
CHECK  CMPM.B      Compare the two memory
       (A2)+,(A3)+ locations
BEQ    CHECK       Keep checking until different
```

The total program occupies only seven words.

A further point is that the CMPM instruction is only allowed with the post-increment mode of addressing; so, if you require any other form then one operand will have to be loaded into a data register first and the CMP instruction used instead.



Change Of Status
The effects of the DIV, MUL and BCD instructions on the status register are shown here. It is important to note that in some cases the absence of a condition does not result in the clearing of the appropriate flag, which will be left unchanged and may, if tested, give a false reading (left over from a previous operation)

Two very important simple arithmetic instructions are NEG and EXT. The NEG or negate instruction subtracts the operand data register from 0 — that is, it forms the two's complement negative value of the data register contents (no other addressing modes are allowed). So, for example, where D0 = 1111 1010 (-6 decimal) and we execute NEG.B D0, then D0 will contain 0000 0110 (6 decimal). Typical uses of this instruction would include forming the absolute value of a data operand by first testing for a negative value and then negating it. There is also an extended form of

this instruction, which includes the X bit, and this is called NEGX.

The EXT or extend instruction is used to extend the sign bit of the data operand into the next larger operand size. Thus, executing EXT.W D0 where D0 = 1111 0101 (FA in hex) would give D0 = FFFA (hex). This instruction is useful when working on multiple-precision numbers, particularly when larger data operands are involved, such as with multiply and divide instructions.

INTERPRETING BIT PATTERNS

Before moving on to discuss more complicated arithmetic instructions, we need first to consider the use of binary bit patterns in a byte data operand. For example, where D0=1001 0110, we can consider this as an integer with a value of −106 (remember that the value is given by inverting and adding one). However, if the number is unsigned and the whole binary range is taken up with positive integers, then the value would be 96 hex, which is equivalent to 150 in decimal (9×16 + 6×1). Unsigned numbers are useful if we know that we only require a large positive range. For example, the address range of a computer could be considered a range of positive unsigned numbers, and provided we don't operate on these numbers with two's complement operators, then all should be well.

There is yet a third interpretation of the bit pattern given; a system called BCD or binary coded decimal. This is a very convenient data code in which each group of four binary digits is considered to be the binary code for one decimal digit. So our example (D0=1001 0110) would give the BCD value of 96 (1001 = 9 and 0110 = 6).

There are three important points to note about this code. First of all, how easy it is to convert even very large numbers from decimal to BCD, or vice versa. For example, 9,631 decimal is 1001 0110 0011 0001 BCD — a conversion which is clearly simple to make.

The second thing to note is the ease with which we can define the precision of numbers coded in this way. Thirdly, you may have noticed that since we are encoding the decimal digits 0 to 9, there are certain codes that are illegal — that is, any code in the range 1010 (10 decimal) to 1111 (15 decimal).

The importance of these points will become clear when we consider the variety of arithmetic instructions available on the 68000. Let's take this further by first looking at the binary multiplication operation. If we multiply two 16-bit operands together, the resulting bit pattern could be 32 bits long. You can check this for yourself, but meanwhile for a word length of n, the product of the largest number, $2^{(n-1)}$, will be $2^{2(n-1)}$, or twice the original word length.

For the binary multiply instruction, we therefore have two 16-bit operands that give a 32-bit result. Since we can work on signed and unsigned numbers, there are two separate instructions — MULU (multiply unsigned) and MULS (multiply signed). Both instructions multiply the two operands together and produce a 32-bit result in the destination data register. Note again that only data addressing modes are allowed for the source operand.

So, for example, MULU #20,D0 for D0 = XXXX 0003 (the X meaning either 0 or 1) would give D0 = 0000 003C after execution. Notice that the whole 32-bit data register is used even though it's not required for this particular example.

Similarly, for D0 = XXXX FFFF, MULU #$10,D0 would produce a result of D0 = 000F FFF0, and a MULS would produce D0 = FFFF FFF0. This is because the result is sign extended to the full 32 bits. Notice that you can check these results easily because the multiplier of 10 hex is equivalent to a left shift of 4, or a multiplication by 16.

A final point on the multiply instruction concerns the condition codes. The N and Z flags are set according to the result, but the V and C bits are set to 0. Although with word operands and long word results we cannot get overflow, it would have been nice to know if the word result had overflowed — then we would easily know if the result could be truncated to a word if necessary.

Let's have a look at a typical small program segment that uses the MULS instruction. Suppose we wanted to convert ASCII characters representing decimal numbers into a binary word. The first of the necessary operations is to convert the input character into binary and then add this bit pattern into a binary accumulator. Before the addition can be done, however, we have to make sure that any previous input is multiplied by 10 (since this is decimal input). So, for example, a possible program might be:

```
SUB.B   #'0',D0     Form the binary of the decimal
                    character
MULS    #10,D5      Multiply old sum by 10
                    decimal
ADD.W   D0,D5       Add in new value
```

In this example the input character is in D0 and the new binary word representing the input characters in binary is in D5.

THE DIVIDE INSTRUCTION

Let's now look at the divide instruction — DIVU for unsigned and DIVS for signed numbers. Both instructions take the 32-bit integer in the destination data register and divide it by the 16-bit source operand. The result is held in the destination data register in the lower 16 bits and any remainder is held in the most significant 16 bits. For instance, where D0 = 0000 0005 (hex):

```
DIVU #3,D0
```

would give D0 = 0002 0001 (1, with remainder 2). Notice that the destination operand is a full 32 bits so a long word clear, or a long sign extended EXT.L, may be necessary first. Since it is possible for overflow to occur (dividing a full 32-bit word by one, for example, will certainly give more than a 16-bit answer), the overflow bit will occasionally have to be taken into consideration.

A simple algorithm for averaging all the numbers in a wordlist terminated by a zero might look like:

```
                ORG $1000
        LEA     LIST1,A0        set up address
POINTER
        CLR.L   D0              clear total
        CLR.L   D1              and loop count
LOOP    ADDQ    #1,D1           count number of
                                words
        ADD.W   (A0)+,D0        and sum
        TST     (A0)            check for end of list
        BNE     LOOP
        DIVU    D1,D0           divide sum by
                                count
        TRAP    0
LIST1   DC.W    1, 2, 3, 4, 0
```

Notice that long word clears have been used in the initialisation part of the program because the sum will be treated as a full 32-bit operand. A count is kept in D1 and the word pointed to be A0 is added to D0 with post-increment addressing. The next word in LIST1 is then tested with the TST instruction. This instruction merely sets the condition codes ready for the BNE instruction — it does not alter any operand.

The TST instruction is necessary because the previous ADD instruction will set the condition codes depending on the result of adding the data operands into D0 — not on the value of the data loaded by the pointer A0. The BNE (Branch if Not Equal to zero) instruction will cause a branch back to LOOP until the next item in the list is zero. When this happens, D0 will contain the sum and the number of non-zero elements (the data count) will be in D1.

When the loop has been executed four times, the values of the data registers will be:

```
D0 = 0000 000A (or 10 decimal)
D1 = 0000 0004
```

So after the DIVU instruction has been executed, D0 will contain 0002 0002 (10÷4 = 2, remainder 2).

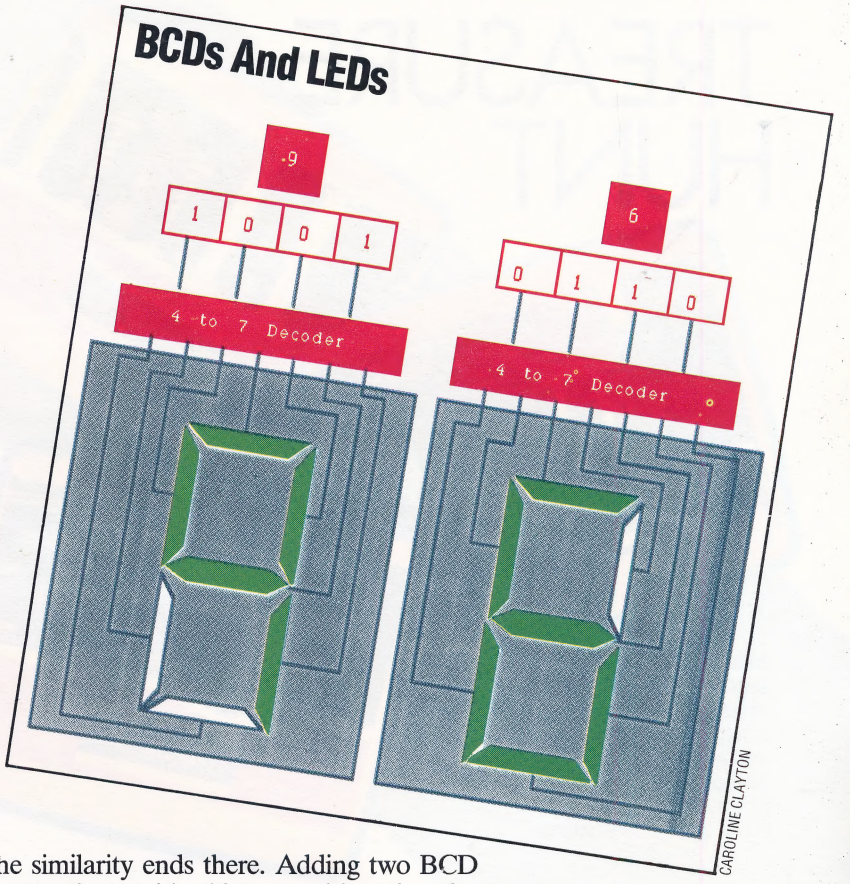
A final point on the divide instructions is that a divide by zero will cause a 'trap' (a software interrupt into the system monitor), since infinity certainly cannot be represented in 16-bits! If you don't want this trap, a simple test on the divisor for zero is all that is required. For example:

```
TST     D1
BEQ     ERROR
DIVU    D1,D0
```

THE BCD INSTRUCTIONS

We have looked at the convenience of BCD for decimal representation. Note, however, that one legal BCD digit corresponds to a hex digit (4 = 0100 binary = 4 hex = 4 BCD), so setting BCD constants will be the same as setting hex constants. For instance:

```
MOVE.B #$54,D0      sets BCD 54 in D0
```



But the similarity ends there. Adding two BCD digits together with binary arithmetic, for example, gives the wrong answer:

```
0100 1001      (BCD 49)    add binary
0000 0001      (BCD 1)
-----------
0100 1010
```

This gives an illegal BCD digit for the least significant BCD digit. This shouldn't cause concern, however, because the 68000 has a set of BCD instructions for add (ABCD), subtract (SBCD) and negate (NBCD).

We'll confine our look at the BCD instructions to ABCD, which adds the source byte (two BCD digits) to the destination byte, with the X bit, producing the sum in the destination. The only addressing modes allowed are data register pairs and pre-decremented address register pairs. So, for example, if D0 = 44 BCD and D1 = 01 BCD, then after:

```
ABCD D0,D1
```

D1 will be 45 BCD. If the X bit in the SR was set (equivalent to BCD carry in), the answer would be 46. Similarly, if we add 1 to 99 in D0, the answer would be 00 with the bit set in the status register. You can see from this that although we are limited to byte operands, we can easily extend the precision of our computation by using the X bit to carry over into more significant byte components.

The condition codes X, C and Z are set for all the BCD instructions. Note, however, that the NBCD instruction (Negate BCD) allows data alterable destination modes rather than just data or pre-decremented pairs.

BCD Precision
As well as operating on signed and unsigned integers, the 68000 has a set of instructions for the manipulation of data held in BCD (binary coded decimal) format. This is particularly useful where total arithmetic precision is required. There are also benefits in other applications — for example, in driving seven-segment LED displays. The hardware decoding of BCD data using a four-to-seven line decoder, as shown here, is easier than decoding binary numbers

TREASURE HUNT

Buried Treasure
Softek's Quo Vadis is a graphics-based adventure with a real payoff. By moving around the locations in the adventure and solving three riddles you can pick up clues to the location of a buried gold and silver sceptre

The chance to win an expensive prize by solving a puzzle posed by an arcade adventure game has lured many a player into these mysterious worlds. We look at one of the most popular of these games, Quo Vadis, which besides having 1,024 screens, offers a gold and silver sceptre as the prize.

The concept of the arcade-adventure, which combines elements of traditional arcade and adventure programs, has been given an added twist by a number of software companies who draw on another tradition that has become popular since the publication of the book *Masquerade* by Kit Williams. This book presented readers with a number of visual and textual clues to the real location of a golden hare, buried somewhere in the English countryside. First to introducê this idea to the computer software market was Automata with its game Pimania, which offered a golden sundial as the prize. Then came Hare Raiser, a computer game similar in concept to *Masquerade* and even offering the identical prize, which was purchased from the original finder and then re-hidden. Other examples include the adventure game Eureka, which offered a prize of £30,000 for the first correct solution, and Firebird Software's prize of a Porsche car for the winner of one of their games.

Softek followed this tradition of offering prizes with their arcade adventurê Quo Vadis, which offered the first person to solve the riddles and escape from the game a gold and silver sceptre. Released in the summer of 1984, an added sensation was caused by the scope of the program, which offered a playing area covering 1,024 different screens.

The game is an arcade-adventure in which you take the role of a knight in search of a sceptre hidden somewhere in a complex system of underground caverns. It's this subterranean maze that occupies the 1,024 screens. Furthermore, the system contains 118 caves, each of which is bigger than the total playing area of many normal arcade adventures.

Moving through the caverns is a question of jumping from one rocky ledge to another, or sometimes climbing up or down ropes. You must, of course, avoid various hazards, such as the lava pits at the bottom of caves which are fatal if fallen into. Combined with these characteristics of the platform game are various elements of the traditional action game. The caverns are inhabited by 38 different kinds of monsters, who attack as the knight enters a cave, and the joystick is used to fire in any one of eight directions.

If the knight is hit by the monsters, he loses energy points and when these have fallen from 100 to zero, the game ends. He can, however, replenish his energy from treasure chests found in certain locations. In some caverns, riddles written on the walls will provide clues that may help you find the sceptre.

It's not just the sheer size of Quo Vadis that's impressive. Instead of the game consisting of separate screens that flip over into another, the picture scrolls with great smoothness to reveal a little more of the surrounding landscape. In this way, the impression is given of slowly penetrating a realistic and mysterious world.

Given the immensity of Quo Vadis, it was inevitable that something else in the game would have to suffer, and it's true to say that the graphics and sound are fairly average. But the graphics become more atmospheric the longer the game is played, and the bright torches, guttering candles, dark stairways and slimy walls all add considerably to the subterranean mood.

The fact that such a large game is capable of being held in the Commodore 64's memory is attributable to the very powerful code compaction techniques used in the programming. Only six bytes are used for each screen, thereby leaving enough space for the game's routines.

Some months after the game's release, Softek brought out on disk a Quo Vadis Generator for those not satisfied with the 1,024 screens already included. The company claims that by using this a million different screens are now available.

Quo Vadis: For the Commodore 64
Price: £9.95, disk; £11.95, cassette
Publishers: Softek International, 12/13 Henrietta St., Covent Garden, London WC2E 8LH
Format: Cassette or disk
Joystick: Required

Motorola 68000 Instruction Set

Here, courtesy of Motorola Inc, we present the first of two sheets which give details of the 68000's instruction set

Mnemonic	Description	Operation	Condition Codes				
			X	N	Z	V	C
ABCD	Add Decimal with Extend	(Destination)$_{10}$ + (Source)$_{10}$ + X → Destination	*	U	*	U	*
ADD	Add Binary	(Destination) + (Source) → Destination	*	*	*	*	*
ADDA	Add Address	(Destination) + (Source) → Destination	–	–	–	–	–
ADDI	Add Immediate	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDQ	Add Quick	(Destination) + Immediate Data → Destination	*	*	*	*	*
ADDX	Add Extended	(Destination) + (Source) + X → Destination	*	*	*	*	*
AND	AND Logical	(Destination) Λ (Source) → Destination	–	*	*	0	0
ANDI	AND Immediate	(Destination) Λ Immediate Data → Destination	–	*	*	0	0
ANDI to CCR	AND Immediate to Condition Codes	(Source) Λ CCR → CCR	*	*	*	*	*
ANDI to SR	AND Immediate to Status Register	(Source) Λ SR → SR	*	*	*	*	*
ASL, ASR	Arithmetic Shift	(Destination) Shifted by <count> → Destination	*	*	*	*	*
Bcc	Branch Conditionally	If cc then PC + d → PC	–	–	–	–	–
BCHG	Test a Bit and Change	~(<bit number>) OF Destination → Z ~(<bit number>) OF Destination → <bit number> OF Destination	–	–	*	–	–
BCLR	Test a Bit and Clear	~(<bit number>) OF Destination → Z 0 → <bit number> → OF Destination	–	–	*	–	–
BRA	Branch Always	PC + d → PC	–	–	–	–	–
BSET	Test a Bit and Set	~(<bit number>) OF Destination → Z 1 → <bit number> OF Destination	–	–	*	–	–
BSR	Branch to Subroutine	PC → – (SP); PC + d → PC	–	–	–	–	–
BTST	Test a Bit	~(<bit number>) OF Destination → Z	–	–	*	–	–
CHK	Check Register Against Bounds	If Dn <0 or Dn> (<ea>) then TRAP	–	*	U	U	U
CLR	Clear an Operand	0 → Destination	–	0	1	0	0
CMP	Compare	(Destination) – (Source)	–	*	*	*	*
CMPA	Compare Address	(Destination) – (Source)	–	*	*	*	*
CMPI	Compare Immediate	(Destination) – Immediate Data	–	*	*	*	*
CMPM	Compare Memory	(Destination) – (Source)	–	*	*	*	*
DBcc	Test Condition, Decrement and Branch	If ~ cc then Dn – 1 → Dn; if Dn ≠ – 1 then PC + d → PC	–	–	–	–	–
DIVS	Signed Divide	(Destination)/(Source) → Destination	–	*	*	*	0
DIVU	Unsigned Divide	(Destination)/(Source) → Destination	–	*	*	*	0
EOR	Exclusive OR Logical	(Destination) ⊕ (Source) → Destination	–	*	*	0	0
EORI	Exclusive OR Immediate	(Destination) ⊕ Immediate Data → Destination	–	*	*	0	0
EORI to CCR	Exclusive OR Immediate to Condition Codes	(Source) ⊕ CCR → CCR	*	*	*	*	*
EORI to SR	Exclusive OR Immediate to Status Register	(Source) ⊕ SR → SR	*	*	*	*	*
EXG	Exchange Register	Rx ↔ Ry	–	–	–	–	–
EXT	Sign Extend	(Destination) Sign-Extended → Destination	–	*	*	0	0
JMP	Jump	Destination → PC	–	–	–	–	–
JSR	Jump to Subroutine	PC → – (SP); Destination → PC	–	–	–	–	–
LEA	Load Effective Address	<ea> → An	–	–	–	–	–
LINK	Link and Allocate	An → – (SP); SP → An; SP + Displacement → SP	–	–	–	–	–
LSL, LSR	Logical Shift	(Destination) Shifted by <count> → Destination	*	*	*	0	*
MOVE	Move Data from Source to Destination	(Source) → Destination	–	*	*	0	0
MOVE to CCR	Move to Condition Code	(Source) → CCR	*	*	*	*	*
MOVE to SR	Move to the Status Register	(Source) → SR	*	*	*	*	*

Λ logical AND
V logical OR
⊕ logical exclusive OR
~ logical complement

* affected
– unaffected
0 cleared
1 set
U undefined


© SHAKIR HUSSAIN